**РЕФЕРАТ**

Розробка веб-сервісу для автоматизації створення та зберігання тестів Testerday // Курсова робота // Рокош Михайло Михайлович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, кафедра комп’ютерних наук, група СН-41 // Тернопіль, 2015 // С. – \_, рисунків – \_, таблиць – \_, джерел – \_.

Ключові слова: METEOR, JAVASCRIPT, MONGODB, HTML, HAML, HTML, СЕРВІС, СSS.

В курсовій роботі розглянуто основні етапи розробки програмного забезпечення, програмні засоби для його створення, вимоги до розробки програмного забезпечення.

Отримані результати – спроектовано, розроблено та протестовано програмне забезпечення.

Основні конструктивні, технологічні і техніко-експлуатаційні характеристики – для користування сервісом необхідно мати персональний комп’ютер підключений до мережі Інтернет.

Рекомендації по впровадженню – створена система може бути розширена шляхом додавання більшої кількості інструментів.

**ЗМІСТ**

[ВСТУП 6](#_Toc420285203)

[1 ОБГРУНТУВАННЯ ВИБРАНИХ ЗАСОБІВ РОЗРОБКИ 8](#_Toc420285204)

[1.1 Обгрунтування вибору мови програмування та фреймворку 8](#_Toc420285205)

[1.2 Обгрунтування вибору середовища розробки 10](#_Toc420285206)

[1.3 Обгрнутування вибору СКБД 11](#_Toc420285207)

[2 ТЕХНІЧНЕ ЗАВДАННЯ НА ВЕБ-СЕРВІС TESTERDAY 13](#_Toc420285208)

[2.1 Загальні відомості 13](#_Toc420285209)

[2.2 Призначення розробки 13](#_Toc420285210)

[2.3 Характеристика об’єкту автоматизації 13](#_Toc420285211)

[2.4 Вимоги до системи 13](#_Toc420285212)

[2.5 Склад і зміст робіт зі створення системи 14](#_Toc420285213)

[2.6 Порядок контролю і приймання системи 14](#_Toc420285214)

[2.7 Вимоги до документування 15](#_Toc420285215)

[3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 17](#_Toc420285216)

[3.1 Діаграма прецидентів 17](#_Toc420285217)

[3.2 Діаграма потоків даних 18](#_Toc420285218)

[3.3 Діаграма класів 19](#_Toc420285219)

[4 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 20](#_Toc420285220)

[4.1 Розробка веб-сайту 20](#_Toc420285221)

[4.2 Постановка завдання 20](#_Toc420285222)

[4.3 Розгортання та перенесення веб-сервісу на віддалений сервер 21](#_Toc420285223)

[4.4 Створення домашньої сторінки 21](#_Toc420285224)

[Рисунок. 4.2 – Сторінка авторизації на сервіс 22](#_Toc420285225)

[4.5 Тестування сайту 23](#_Toc420285226)

[5 ОПИС ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ 25](#_Toc420285227)

[5.1 Загальні відомості про програму 25](#_Toc420285228)

[5.2 Функціональне призначення 25](#_Toc420285229)

[5.3 Опис логічної структури 25](#_Toc420285230)

[5.4 Використовувані технічні засоби 26](#_Toc420285231)

[5.5 Виклик і завантаження 26](#_Toc420285232)

[5.6 Вхідні дані 26](#_Toc420285233)

[5.7 Вихідні дані 27](#_Toc420285234)

[ВИСНОВОК 28](#_Toc420285235)

[СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ 29](#_Toc420285236)

[ДОДАТКИ](#_Toc420285237)

# ВСТУП

Об'єктивною реальністю сьогодення є широке впровадження у сфери життєдіяльності особи, суспільства та держави сучасних інформаційних технологій, розгортання на їх основі локальних і глобальних інформаційних систем, призначених для прискорення обміну інформацією та доступу до різноманітних інформаційних ресурсів.

Інтернет являє собою надзвичайно привабливий засіб комунікації, що може пояснити його безпрецедентний розвиток. Новітні засоби зв'язку дозволили об'єднати розрізнені комунікаційні системи у глобальну мережу. Завдяки цьому людина отримала можливість обмінюватися інформацією в межах всієї планети, не залежно від кордонів і відстаней. Саме Інтернет є однією з ознак переходу суспільства від індустріальної стадії розвитку до інформаційної.

Стрімкий розвиток мережевих інформаційних технологій, окрім помітного зниження тимчасових і просторових бар'єрів в розповсюдженні інформації, відкрив нові перспективи у сфері освіти. Можна з упевненістю стверджувати, що в сучасному світі має місце тенденція злиття освітніх і інформаційних технологій і формування на цій основі принципово нових інтегрованих технологій навчання, заснованих, зокрема, на Інтернет-технологіях.

Інформаційні технології - процес, що використовує сукупність засобів і методів збору, обробки і передачі даних, для отримання інформації нової якості про стан об'єкту, процесу або явища.

Згідно визначенню, прийнятому ЮНЕСКО, інформаційна технологія - це комплекс взаємозв'язаних, наукових, технологічних, інженерних дисциплін, що вивчають методи ефективної організації праці людей, зайнятих обробкою і зберіганням інформації; обчислювальну техніку і методи організації і взаємодії з людьми і виробничим устаткуванням, їх практичні додатки, а також пов'язані зі всім цим соціальні, економічні і культурні проблеми.

Самі інформаційні технології вимагають складної підготовки, великих первинних витрат і наукоємкої техніки. Їх введення повинне починатися із створення математичного забезпечення, формування інформаційних потоків в системах підготовки фахівців.

Одним з перспективніших напрямів розвитку системи освіти є широке використання сучасних телекомунікаційних, інформаційних і комп'ютерних технологій, в першу чергу - технологій глобальної мережі Інтернет.

Зручність і гнучкість гіпертекстового представлення матеріалу, оперативний доступ до інформації, розташованої в різних регіонах і країнах, висока оперативність оновлення і інші переваги інтернет-технологій дозволили достатньо швидко упровадити їх в практику багатьох навчальних закладах.

Testerday.com дозволяє зареєстрованим користувачам створювати приватні чи публічні тести за допомогою інтуїтивно-зрозумілого веб-інтерфейсу. Таким чином, даний сервіс буде корисним не тільки навчальним закладам, а й компаніям, які зацікавлені в наймі на роботу нових працівників.

# 1 ОБГРУНТУВАННЯ ВИБРАНИХ ЗАСОБІВ РОЗРОБКИ

* 1. **Обгрунтування вибору мови програмування та фреймворку**

Javascript - скриптова мова, призначена для створення інтерактивних веб-сторінок, проте на цьому сфера її застосування не закінчується. Офмфіскшзе активно використовують при розробці серверної частини веб-додатку чи навіть ігор.

У javascript є свій стандарт: ECMAScript, специфікація якого знаходиться на сайті в розділі «стандарт мови».

За допомогою javascript можна виконувати наступні дії:

* змінювати сторінку, писати на ній текст, додавати і видаляти теги, міняти стилі елементів;
* реагувати на події: скрипт може чекати, коли щось станеться (клік, закінчення завантаження сторінки) і реагувати на це виконанням функції;
* виконувати запити до сервера і завантажувати дані без перезавантаження сторінки. Це іноді називають "AJAX";
* встановлювати і зчитувати cookie, валідувати дані, виводити повідомлення і багато іншого.

До сильної сторони даної мови належить повна підтримка браузерами. Наприклад, такі технології як ActiveX, VBScript, XUL - підтримуються не в кожному браузері. Такі технології як Flash, Silverlight, Java - не повністю інтегровані з браузером, працюють в своєму оточенні.

Дана мова програмування не позбавлена недоліків, таких, як, наприклад, нестрога типізація, що є причиною неявного переведення типів. Однак усі недоліки перекриваються поширеністю та кросбраузерністю мови.

Для більш продуктивного і простого процесу розробки було вибрано фреймворк MeteorJS, в якому використовується мова javascript як на клієнті, так і на сервері, а також в роботі з базою даних, що дозволяє виконувати один і той самий код на фронт-енді та бек-енді одночасно.

Meteor – це насамперед дві речі: бібліотека пакетів та утиліта командного рядка.

Ядро Meteor складається з дюжини пакетів, які використовуються в більшості додатків (наприклад пакет webapp, який обробляє вхідні HTTP-підключення, і templating, який дозволяє писати HTML-шаблони, автоматично оновлювані при зміні даних). Також існують додаткові пакети, наприклад email, який дозволяє додатку посилати email-и, або серія пакетів Meteor Accounts (account-password, accounts-facebook, accounts-ui та інші) які надають повноцінну систему облікових записів для вашої програми. І крім цих "офіційних" пакетів, в Atmosphere присутні сотні інших, написаних співтовариством.

Утиліта командного рядка «meteor» - це утиліта для побудови додатку, аналогічна make, rake, або без-інтерфейсним частинам Visual Studio. Вона збирає воєдино всі вихідні коди і статичні файли додатку, виконує всі необхідні кроки збірки (такі, як компіляція CoffeeScript, мініфікація CSS, збірка npm-модулів і генерація source maps), завантажує всі модулі, використовувані у вашому додатку і створює з нього автономний пакет, готовий до використання. У режимі розробки вона робить все це "на льоту", тобто кожен раз, коли ви вносите будь-які зміни у файл, ви відразу ж бачите це в браузері. Вона дуже проста у використанні і до того ж легко розширюється.

Основна ідея пакетної системи Meteor полягає в тому, що все має працювати однаково як в браузері, так і на сервері (звичайно, якщо це має сенс: браузери не вміють посилати email-и, а сервери - відловлювати події мишки).

Сім Принципів Meteor:

* Data on the wire. На клієнт відправляється не згенерований сервером HTML, а дані в форматі EJSON.
* Одна мова. І клієнтська і серверна частини пишеться на мові програмування JavaScript.
* База даних скрізь. Однакове API для доступу до бази даних як з сервера, так і клієнта.
* Компенсація затримок. Попередня симуляція виконання серверних методів на клієнті для гарантування відсутності затримки при зверненні до бази даних.
* Реактивність всюди. Всі шари додатка, починаючи з бази даних і закінчуючи шаблонами, підтримують подієво-орієнтований інтерфейс.
* Використання екосистеми. У Meteor відкритий вихідний код і він, швидше, інтегрується в інші opensource-фреймворки та засоби розробки, ніж замінює їх.
* Простота рівна продуктивності. Кращий спосіб змусити щось здаватися простим - зробити це дійсно простим. Це стосується і API пакетів.
  1. **Обгрунтування вибору середовища розробки**

Під час розробки Testerday використовувався текстовий редактор Sublime Text 2 та VIM.

Sublime Text 2 – швидкий [кросплатформенний](http://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%BD%D1%96%D1%81%D1%82%D1%8C) редактор файлів. Підтримує плагіни, розроблені за допомогою мови програмування [Python](http://uk.wikipedia.org/wiki/Python). Велика кількість плаганів, які в рази покращують продуктивність і швидкодію роботи програмістів, зручний і зрозумілий в користуванні. Недоліки редактора те, що він не є безкоштовним.

Vim — [текстовий редактор](http://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80), створений на основі старішого [vi](http://uk.wikipedia.org/wiki/Vi). Нині це один з найпотужніших текстових редакторівз повною свободою налаштування та автоматизації й великою кількістю створених завдяки цьому розширень та надбудов.

* 1. **Обгрнутування вибору СКБД**

Напевне найголовнішим і найвідповідальнішим етапом під час розробки веб-сайту є проектування і розробка бази даних.

Основою бази даних є модель даних — фіксована система понять і правил для представлення даних структури, стану і динаміки проблемної області в базі даних. У різний час послідовне застосування одержували ієрархічна, мережна і реляційна моделі даних. У наш час усе більшого поширення набуває об'єктно-орієнтований підхід до організації баз даних.

Система управління базами даних — це сукупність програмних і лінгвістичних засобів загального або спеціального призначення, що забезпечують керування створенням і використанням баз даних.

Реляційна модель орієнтована на представлення даних у вигляді двовимірних таблиць. Таке уявлення зручно для користувачів. Але реляційні бази даних не можуть справлятися з навантаженнями актуальними в наш час.

Три основних проблеми РСУБД:

* горизонтальна масштабування при великих обсягах даних;
* продуктивність кожного окремого сервера;
* негнучкий дизайн логічної структури;

Все це змушує розробників використовувати альтернативи реляційних баз даних, які відносяться до сімейства NoSQL.

Термін NoSQL був придуманий Еріком Евансом. NoSQL в інформатиці - термін, що позначає ряд підходів, проектів, спрямованих на реалізацію моделей баз даних, що мають суттєві відмінності від використовуваних реляційних СУБД з доступом до даних засобами мови SQL.

NoSQL - важливий і корисний інструмент, але він не може вважатися універсальним. Основна мета підходу - розширити можливості БД там, де SQL недостатньо гнучкий, і не витісняти його там, де він справляється зі своїми завданнями. В основі ідеї NoSQL лежать:

* нереляційних модель даних;
* відкритий вихідний код;
* хороша горизонтальна масштабованість;

Характерним представником СКБД сімейства NoSQL є MongoDB — документно-орієнтована СКБД. Вона офіційно підтримується фреймворком Meteor.js, а також є невід’ємною частиною MEAN-стеку. Саме цю СКБД використовує веб-додаток Testerday. Основною причиною вибору MongoDB є відсутність офіційної підтримки інших СКБД. Однак, саме MongoDB найбільше підходить фреймворку MeteorJS, оскільки відповідає вимогам одного з семи принципів – «Одна мова – Javascript».

MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. У MongoDB є вбудовані засоби із забезпечення шардінга (розподіл набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання, в якому відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера в кластер проводиться без зупинки роботи БД простим додаванням нових машин.

# 2 ТЕХНІЧНЕ ЗАВДАННЯ НА ВЕБ-СЕРВІС TESTERDAY

* 1. **Загальні відомості**

2.1.1 Назва завдання: розробка веб-сервісу для створення тестів Testerday.

2.1.2 Виконавець завдання: Рокош Михайло Михайлович, студент групи СН-41 кафедри комп’ютерних наук факультету комп’ютерно-інформаційних систем та програмної інженерії Тернопільського національного технічного університету імені Івана Пулюя.

2.1.3 Термін виконання завдання: 4.03.2015р. – 30.04.2015р.

* 1. **Призначення розробки**

2.2.1 Призначення сервісу.

ПЗ «Testerday» є засобом для автоматизації створення та зберігання тестів.

2.2.2 Цілі створення системи

ПЗ «Testerday» створюється з метою автоматизації створення та зберігання тестів.

Створення ПЗ «Testerday» має метою забезпечення:

* скорочення часу на створення тестових завдань;
* зберігання і перегляд тестових питань онлайн;
  1. **Характеристика об’єкту автоматизації**

Об’єктом автоматизації є система, яка виконує семантичний розбір лекційного матеріалу і генерує тестові питання.

Інформаційна система повинна бути проста для користувача без додаткових навиків.

* 1. **Вимоги до системи**
     1. Вимоги до засобів розробки
  + наявність інтегрованого середовища розробки;
  + наявність засобів, що забезпечують швидке візуальне проектування компонентів прикладного програмного забезпечення.

2.4.2 Мінімальні вимоги до програмної сумісності

Для того щоб користуватися сервісом необхідно мати комп'ютер з доступом в інтернет.

* + 1. Мінімальні вимоги для апаратної сумісності
* 256 Mb оперативної пам’яті;
* вбудована відеокарта;
* браузер;
* процесор з частотою 1.6 ГГц.
  1. **Склад і зміст робіт зі створення системи**

Роботи зі створення сервісу передбачається проводити в два етапи:

* + передпроектне і проектне опрацьовування;
  + реалізація і впровадження програми.

Матеріали робіт оформлюються згідно з існуючими державними стандартами та відповідності до план-графіку проведення робіт:

* + узгодження технічного завдання;
  + узгодження технічних пропозицій та рішень;
  + проведення тестування сервісу;
  + уточнення технічного завдання та доопрацювання технічних рішень;
  + введення сервісу в експлуатацію.
  1. **Порядок контролю і приймання системи**

2.6.1 Види, склад, обсяг і методи випробувань системи та її складових частин

Тестування сервісу обов’язково проводиться на декількох бравзерах.

Для сервісу встановлюються такі основні види випробувань:

* попередні випробування (проводять після налагодження і тестування розробником системи);
* дослідна експлуатація (проводять з метою перевірки експлуатаційних та технічних характеристик програмного забезпечення);
* приймальні випробування (комплексна перевірка) (проводять для остаточного визначення відповідності сервісу вимогам ТЗ, оцінки якості дослідної експлуатації та рішення питання про можливість сервісу на постійні основі).
  1. **Вимоги до документування**

Всі документи оформляються по відповідних правилах, вимогам ГОСТ і ЕСКД, з наступними уточненими вимогами до документів, що готуються в електронній формі:

* набір тексту, формул і вставка малюнків мають бути зроблені в редакторі Word 2010 (розширення \*.docx або \*.rtf);
* формат сторінки – А4: відступи зліва – 25 мм, справа – 15 мм, зверху – 20 мм, знизу 20 мм;
  + шрифт основного тексту – Times New Roman Cyr, розмір шрифту 14 пунктів; міжрядковий інтервал – 1,5;
* слово «Таблиця» і її порядковий номер розташовується в правому верхньому кутку. Назва таблиці розташовується по центру. Посилання на таблицю 1 в тексті – «Табл.1»;
* малюнки мають бути вставлені в текст і відділятися одним порожнім рядком до і після малюнка, підпис до малюнка знизу.

# 3 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## **3.1 Діаграма прецидентів**

Діаграми варіантів використання складають модель варіантів використання (use-cases). Варіант використання – це функціональність системи, яка дозволяє користувачеві отримати якийсь істотний для нього, відчутний та вимірюваний результат. Кожен варіант використання відповідає окремому сервісу, що надається модельованою системою у відповідь на запит користувача, тобто визначає спосіб використання цієї системи.

Варіанти використання найчастіше застосовуються для специфікації зовнішніх вимог до проектованої системи або для специфікації функціональної поведінки вже існуючої системи. Окрім цього, варіанти використання неявно описують типові способи взаємодії користувача з системою, що дозволяють коректно працювати з сервісами, що надаються системою.

Діаграма варіантів використання сервісу представлена на рисунку 3.1.

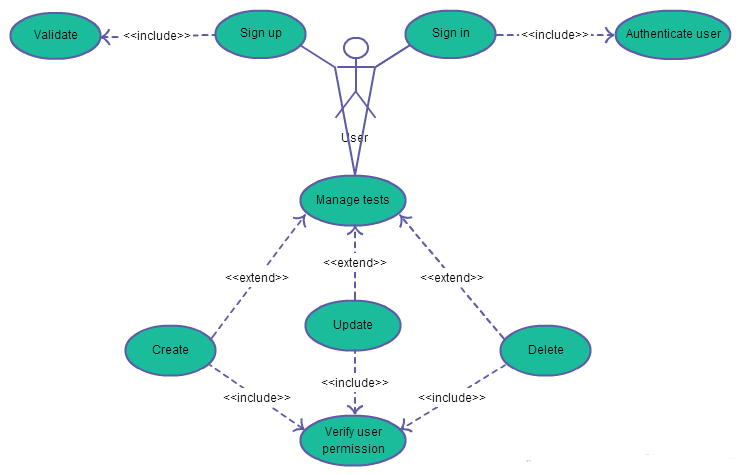


Рисунок 3.1 – Діаграма варіантів використання

## **3.2 Діаграма потоків даних**

Діаграма потоків даних  — модель проектування, графічне представлення «потоків» даних в [інформаційній системі](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних ([структурне проектування](http://uk.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F&action=edit&redlink=1)).

Для розробника вважається звичним cпочатку креслити діаграму потоків даних рівня контексту, завдяки чому буде показано взаємодію системи із зовнішніми модулями. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати розлого розроблювану систему.

На рисунку 3.2 представлено діаграму потоків даних косметичного салону:

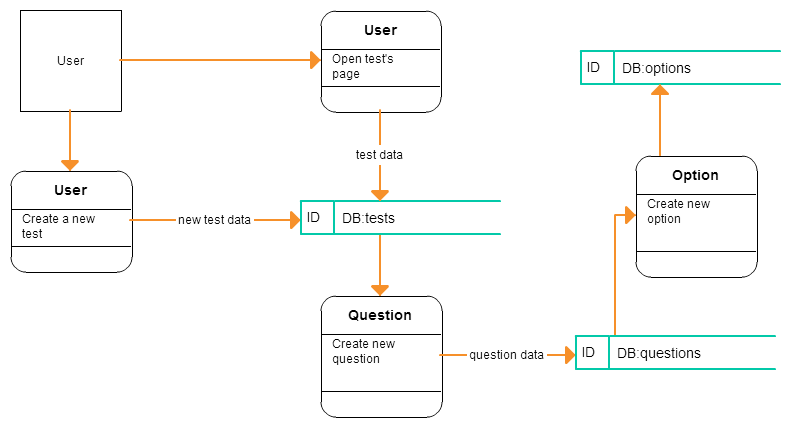


Рисунок 3.2 – Діаграма потоків даних

## **3.3 Діаграма класів**

Діаграма класів призначена для надання статичної структури моделі системи в термінології класів об’єктно-орієнтованого програмування. Діаграма класів відображує різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти й підсистеми, а також описує їхню внутрішню структуру й типи відносин. На даній діаграмі не вказується інформація про часові аспекти функціонування системи.

Клас визначає атрибути і методи набору об’єктів. Всі об’єкти цього класу (екземпляри цього класу) мають спільну поведінку і однаковий набір атрибутів (кожен з об’єктів має свій власний набір значень).

Діаграма класів для даного сервісу містить класи:

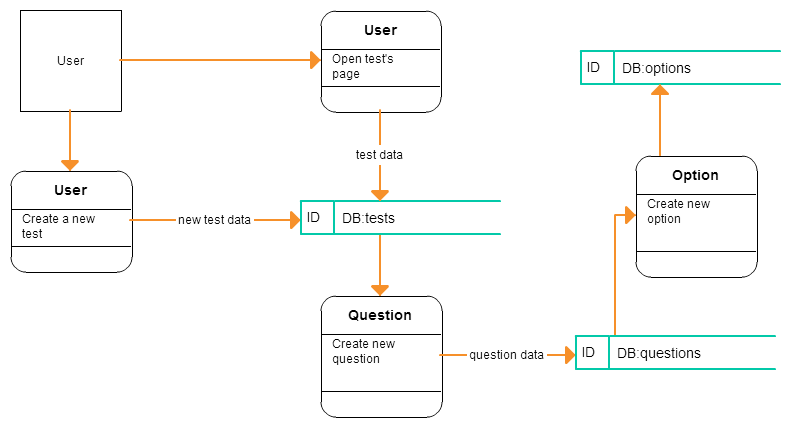


Рисунок 3.3 – Діаграма класів

# 4 РОЗРОБКА І ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

# 4.1 Розробка веб-сайту

Розробка веб-додатку Testerday проходила в декілька етапів. В наступних розділах буде описаний кожен з цих етапів.

Провівши аналіз предметної області, було зроблено висновок, що потреба автоматизації генерації тестів є досить актуальною в наш час. В даному випадку, автоматизація проводиться на базовому рівні, адже інтелектуальна система генерації стладних за структурою питань та відповідей є досить трудомістким процесом.

База даних повинна містити різноманітну інформацію, як, наприклад, спроби здачі тестів з вибраними варіантами відповідей. Чим більше інформації вдається зберегти, тим детільнішу можна буде складати статистику. Сильною стороною документно-орієнтованої СКБД MongoDB є її безструктурність, а це означає, що структурна схема документів питань та тестів може видозмінюватись з часом.

Отже, інформаційна система повинна містити повну інформацію про користувачів, опубліковані тести та інші дані, необхідні для ведення обліку і статистики здачі тестів, зручну навігацію та пошук.

## **4.2 Постановка завдання**

Попередній аналіз предметної області свідчить про необхідність реалізації наступних завдань:

* авторизація користувачів;
* зручне середовище для редагування та створення питань;
* збереження та публікація тестів;
* можливість управління створеними тестами;
* можливість адміністратора переглядати дані всіх колекцій в зручному для нього вигляді;
* можливість адміністратора керувати зареєстрованими в системі користувачами;

Після підсумування вимог можна зробити висновок, що веб-додаток буде мати зручний користуваціький інтерфейс, доступний із будь якого комп’ютера або мобільного пристрою із доступом до інтернету.

## 4.3 Розгортання та перенесення веб-сервісу на віддалений сервер

Використаємо команду Meteor для того, щоб створити місце на тестових серверах Meteor для нашого додатку, скориставшись командою у лістингу 4/1.

Лістинг 4.1 – розгортання додатку на тестових серверах Meteor

$ meteor deploy testerday.meteor.com

Deploying to testerday.meteor.com.

Now serving at http://testerday.meteor.com

Це все що потрібно зробити, щоб запустити простий порожній додаток на віддалених серверах Meteor. Тепер додаток доступний для всіх пристроїв, підключених до інтернету.

Лістинг 4.2 – Завантаження коду в Github репозиторій

$ git init

$ git add —-all

$ git commit -m "first commit"

$ git remote add origin https://github.com/Praxie/Testerday.git

$ git push -u origin master

## **4.4 Створення домашньої сторінки**

Важливим етапом розробки веб-додатку є створення інформативної домашньої сторінки (рисунок 4.1).

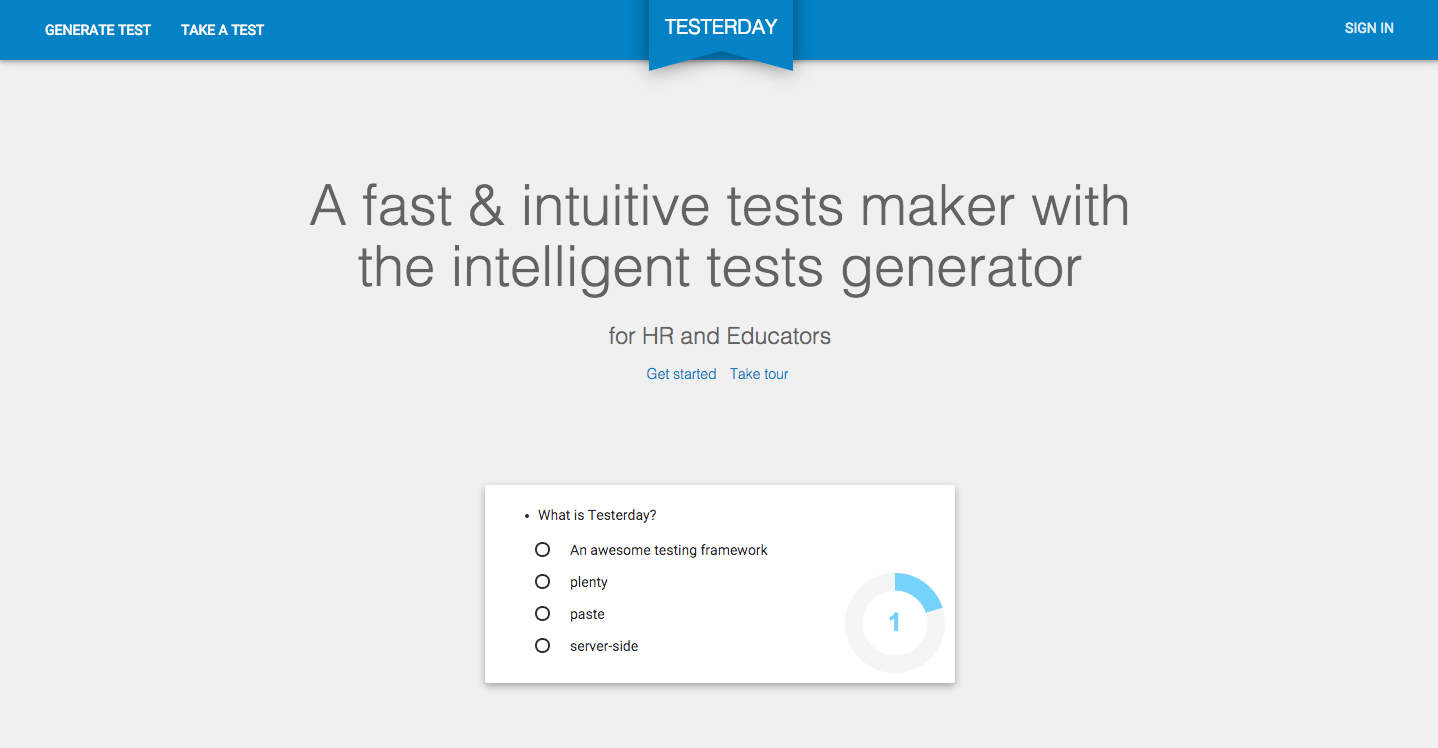
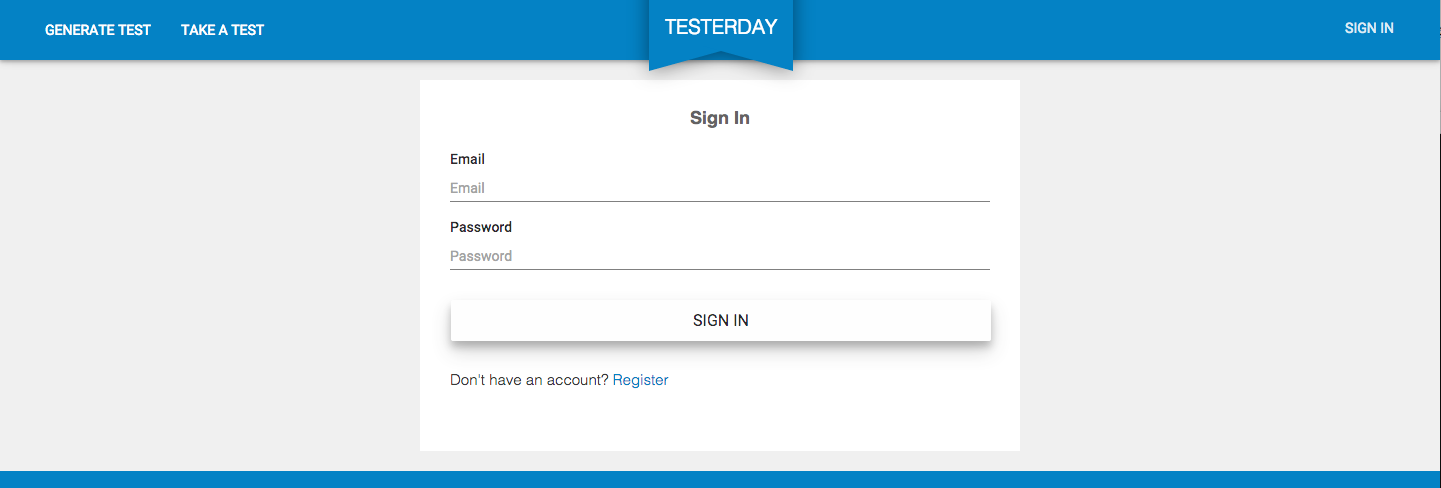


Рисунок. 4.1 – Головна сторінка сервісу

Окрім того було розроблено сторінку авторизації з полями для вводу паролю та адреси електронної пошти (рисунок 4.2).

# Рисунок. 4.2 – Сторінка авторизації на сервіс

# 4.5 Тестування сайту

Даний розділ присвячений тестуванню веб-сайту після завершення його розробки та аналізу результатів тестування, адже потрібно переконатись чи все було правильно зроблено, чи все нормально відображається.

Загалом тестування сайтів проводиться в кілька етапів:

* залучається велика кількість тестерів для проведення тестування на зручність;
* перевіряються посилання;
* перевірка можливості сайту працювати при великих навантаженнях.

Крім цього веб-сайти перевіряються на сумісність з сучасними браузерами. Для цього використовується декілька різних найпопулярніших браузерів, потім порівнюються результати відображення і на результатах робляться висновки щодо покращень на певному браузері, якщо той неправильно щось відображає.

В даному випадку тестування веб-сайту буде проводитись на двох браузерах:

* Google Chrome;
* Safari.

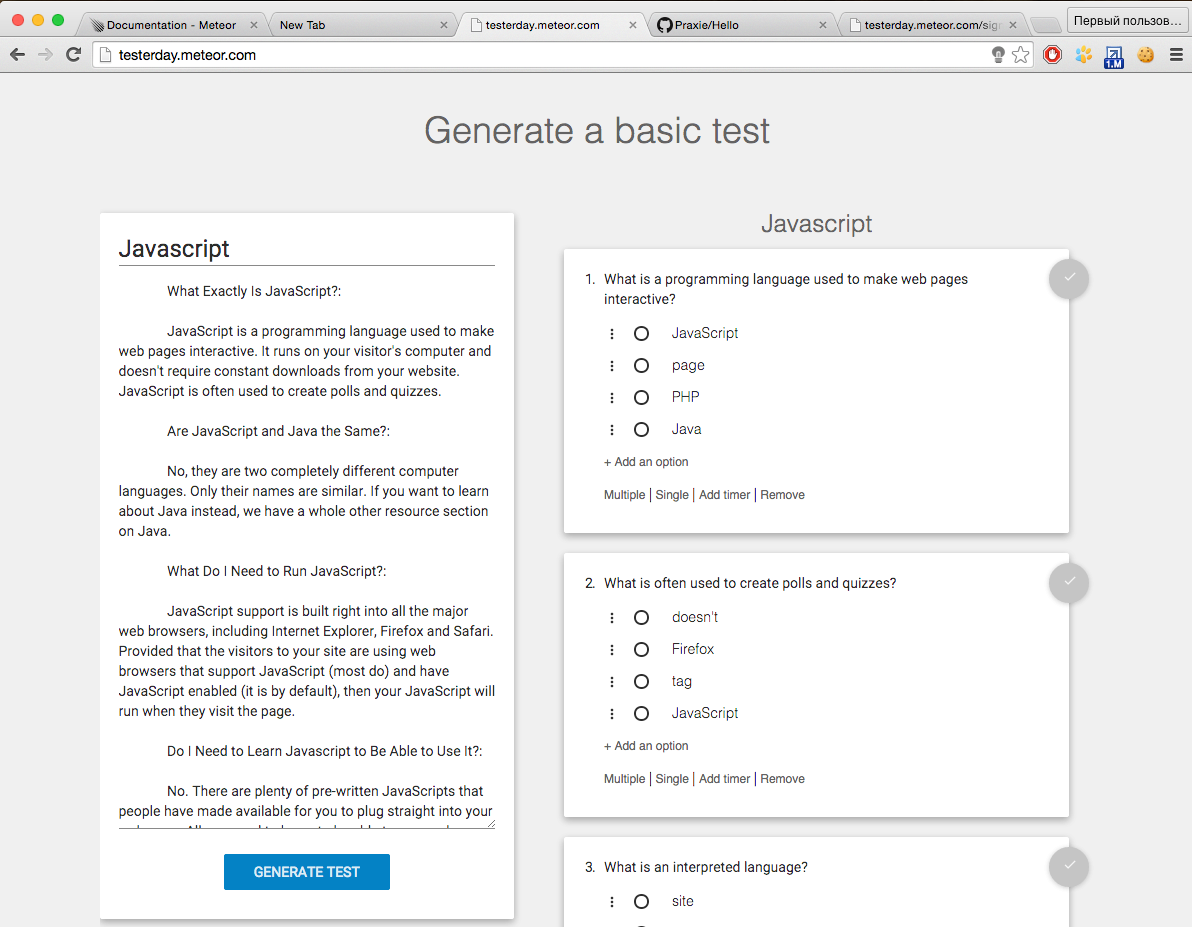


Рисунок 4.4 – Вигляд сайту в Google Chrome

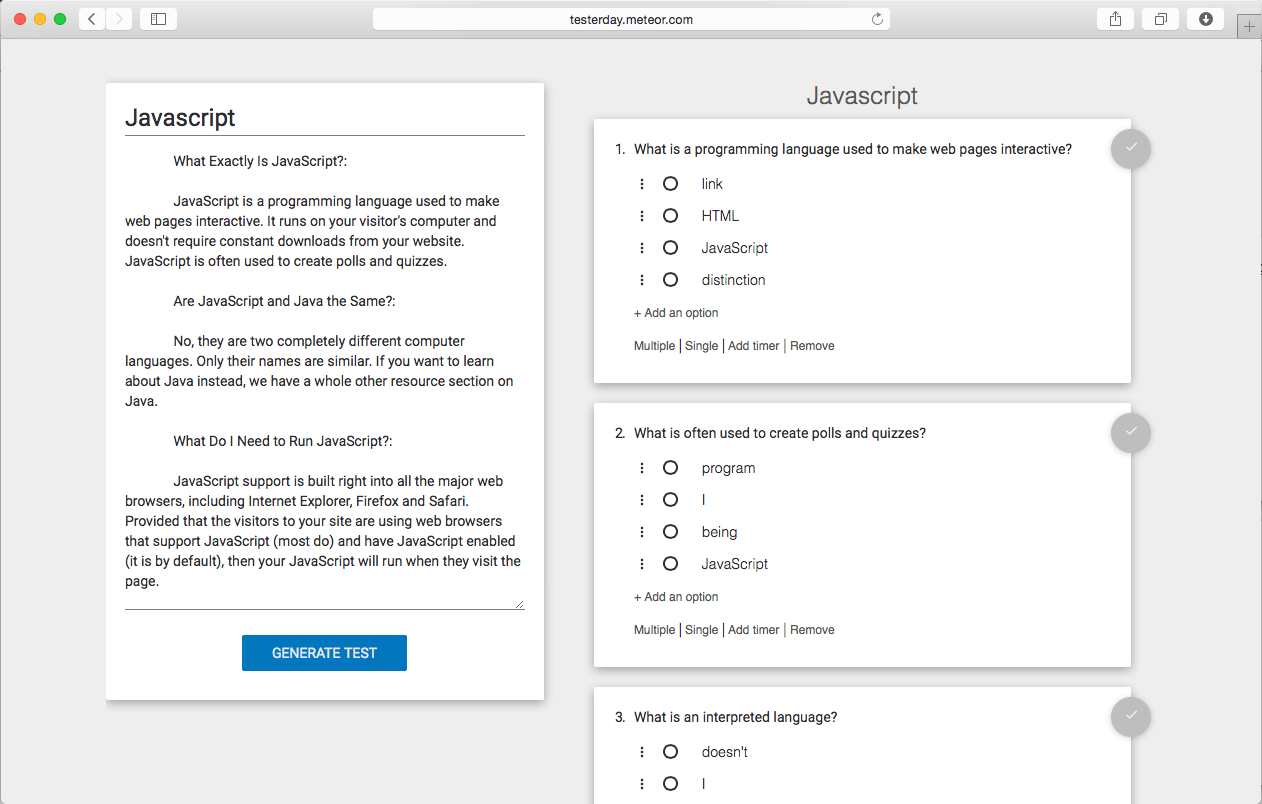


Рисунок 4.3 – Вигляд сайту в Safari

Веб-сайт було протестовано. Судячи по результатах тестування помітних недоліків відображення сайту в жодному браузері не знайдено, а отже даний веб сайт сумісний з сучасними браузерами і працює корректно.

# 5 ОПИС ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

## **5.1 Загальні відомості про програму**

Веб-сервіс Testerday призначений для австоматизації процесу створення та збереження тестів в режимі онлайн.

## **5.2 Функціональне призначення**

Провівши аналіз предметної області, було зроблено висновок, що потреба автоматизації створення та збереження тестів онлайн є досить актуальною в наш час. Враховуючи її потенціал в освітній сфері, було передбачено можливість розширення функціоналу, наприклад, проходження створених тестів іншими користувачами, а також генерація тестових питань та варіантів відповідей на основі текстового матеріалу.

Програма має зручний, інтуїтивно зрозумілий інтерфейс користувача.

## **5.3 Опис логічної структури**

В нашому випадку буде 4 колекцій в базі даних для збереження інформації. Провівши огляд колекцій, можна навести список потрібних моделей та обєкти в предметній області які вони представляють:

* User – користувач системи;
* Test – тест;
* Question – питання, яке відноситься до конкретного тесту;
* Option – варіант відповіді, який відноситься до конкретного питання.

## **5.4 Використовувані технічні засоби**

Сервіс може бути запучений на будь якому комп’ютері з доступом в інтернет, перейшовши по посиланню https://testerday.meteor.com/.

Для роботи з продуктом пропонуються ПК з наступними мінімальними вимогами до апаратної та програмної частини:

* + 256 Mb оперативної пам’яті;
  + вбудована відеокарта;
  + процесор з частотою 1.6 ГГц;
  + браузер.

## **5.5 Виклик і завантаження**

Запуск сервісу здійснюється за допомогою будь якого браузера, перейшовши по посиланню https://testerday.meteor.com/. Після чого потрібно авторизуватися, ввівши унікальний логін та пароль.

## **5.6 Вхідні дані**

Вхідними даними є:

* інформація про користувачів;
* інформація про тести.

Всі дані вводять за допомогою інтуїктивно зрозумілого графічного інтерфейсу з використанням мишки та клавіатури. Також підтримується функція «drag’n’drop» для зміни порядку розташування варіантів відповідей та питань.

## **5.7 Вихідні дані**

Вихідними даними є відображення на екрані попередньо створених тестів з питаннями, відповідного їм текстового контенту, а також інформації про користувачів.

# ВИСНОВОК

Під час роботи було повністю реалізовано поставлену задачу – розробити веб-сервіс для створення та переглядання тестів Testerday.

Проведено дослідження предметної області. На основі результатів розроблена нова структура представлення даних в інформаційних системах, створені моделі даних кожного розділу; наявні менеджери даних, що виконують функцію з обробки та виконання операцій, оптимізовані під поставлені вимоги, створені системи взаємодії з користувачем.

На даний момент програма розташовуєься в github репозиторії і має відкритий код для можливості покращення її інишими розробниками, а також розміщена на стандартному хостингу meteor, який є зручним і безкоштовним, проте з певними обмеженнями в користуванні.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Вікіпедія. The Free Encyclopedia [Електронний ресурс] Meteor (web framework) — 2011. — Режим доступу: [www.en.wikipedia.org/wiki/Meteor\_](http://www.en.wikipedia.org/wiki/Meteor_)(web\_framework) — Дата доступу: 03.06.2014. — Назва з домашньої сторінки інтернету.
2. Documentation - Meteor [Електронний ресурс] Веб сервер — 2013. — Режим доступу: www.docs.meteor.com/#/full/ — Дата доступу: 03.02.2015. — Назва з домашньої сторінки інтернету.
3. Портал знань [Електронний ресурс] Поняття веб сайту — 2011. — Режим доступу: www.znannya.org/1896-ponyattya-veb-saitu-poryadok-vdobrazhennya-stvorennya-veb-saita-u-bu.html — Дата доступу: 07.01.2015. – Назва з домашньої сторінки інтернету.
4. Hochhaus S. Meteor in Action / S. Hochhaus, M. Schoebel. — Manning Publications Co., 2014. — 389 p. — ISBN 978-1430224334.
5. Вікіпедія. The Free Encyclopedia [Електронний ресурс] Система керування вмістом — Режим доступу: http://uk.wikipedia.org/wiki/Система\_керування\_вмістом — Дата доступу: 05.01.2015. — Заголовок з екрану.
6. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению — М.: Изд-во стандартов, 1980. — 4 с.

# ДОДАТКИ

Додаток А

Лістинг «editable\_test\_item.js»

Template**[**getTemplate('editable\_test\_item')**]**.events({

'click .add-question-btn'**:** **function**(e) {

e.preventDefault();

**var** lastRank **=** Blaze.getData($('.question-wrapper:last')**[**0**]**).rank;

**var** questionId **=** Questions.insert({

testId**:** **this**.test.\_id,

question**:** "Question...",

answer**:** "Right option",

type**:** 'radio',

rank**:** lastRank **+** 1,

custom**:** **true**

});

Options.insert({

testId**:** **this**.test.\_id,

questionId**:** questionId,

string**:** 'Right answer',

correct**:** **true**,

rank**:** 1

});

}

});

Template**[**getTemplate('editable\_test\_item')**]**.rendered **=** **function**() {

Sortable.create(questionsList, {

group**:** 'questionsList',

sort**:** **true**,

draggable**:** '.question-wrapper',

animation**:** 150,

ghostClass**:** "question-ghost",

onEnd**:** **function** (evt) {

**var** before, after, newRank;

before **=** evt.item.previousElementSibling;

after **=** evt.item.nextElementSibling;

**if** (evt.newIndex **===** evt.oldIndex) **return**;

**if** (**!**before) {

*//if it was dragged into the first position grab the*

*// next element's data context and subtract one from the rank*

newRank **=** Blaze.getData(after).rank **-** 1;

} **else** **if** (**!**after) {

*//if it was dragged into the last position grab the*

*// previous element's data context and add one to the rank*

newRank **=** Blaze.getData(before).rank **+** 1;

} **else** {

*//else take the average of the two ranks of the previous*

*// and next elements*

newRank **=** (Blaze.getData(after).rank **+**

Blaze.getData(before).rank)**/**2;

}

*//update the dragged Question's rank*

Questions.update({\_id**:** Blaze.getData(evt.item).\_id}, {$set**:** {rank**:** newRank}});

},

});

};

Додаток Б

Лістинг «editable\_question.js»

Template**[**getTemplate('editable\_question')**]**.events({

'click .enable-timer-btn'**:** **function**(e, tmpl) {

**var** self **=** **this**;

Questions.update({\_id**:** **this**.\_id}, {$set**:** {timer**:** 10, promo**:** **true**}}, **function**() {

$(tmpl.firstNode).find('.timer').val(10);

$(tmpl.firstNode).find('.timer').knob({

min**:** 0,

max**:** 100,

width**:** 100,

height**:** 100,

'release' **:** **function** (val) {

Questions.update({\_id**:** self.\_id}, {$set**:** {timer**:** val}});

}

});

});

},

'click .remove-question-btn'**:** **function**(e) {

Questions.remove({\_id**:** **this**.\_id});

},

'dragend .option-wrapper'**:** **function**(e) {

console.log("DRAG END", e);

},

'focusout .question-text'**:** **function**(e, tmpl) {

Questions.update({\_id**:** **this**.\_id}, {$set**:** {question**:** $(e.currentTarget).text()}});

},

'focusout .option-text'**:** **function**(e, tmpl) {

Options.update({\_id**:** **this**.\_id}, {$set**:** {string**:** $(e.currentTarget).text()}});

},

'click .approve-btn'**:** **function**(e, tmpl) {

$(e.currentTarget).toggleClass('btn-unapproved');

$(e.currentTarget).toggleClass('btn-primary');

},

'click .radio-option'**:** **function**(e, tmpl) {

console.log('this', Template.parentData(0)); *// Template.parentData(0) gets current template context. (Otherwise 'this' returns a string - an options***[]** *element)*

console.log('$(e.currentTarget).val()', $(e.currentTarget).val());

**if** (**this**.correct) {

$(e.currentTarget).closest('.radio').addClass('radio-success');

console.log("correct");

} **else** {

$(e.currentTarget).closest('.radio').addClass('radio-danger');

}

},

'click .add-option-btn'**:** **function**(e, tmpl) {

**var** lastRank **=** Blaze.getData($(tmpl.firstNode).find('.option-wrapper:last')**[**0**]**).rank;

Options.insert({

testId**:** **this**.testId,

questionId**:** **this**.\_id,

string**:** 'One more option',

correct**:** **false**,

rank**:** lastRank **+** 1

});

},

'click .enable-checkbox-btn'**:** **function**(e) {

Questions.update({\_id**:** **this**.\_id}, {$set**:** {type**:** 'checkbox'}});

},

'click .enable-radio-btn'**:** **function**(e) {

Questions.update({\_id**:** **this**.\_id}, {$set**:** {type**:** 'radio'}});

}

});

Template**[**getTemplate('editable\_question')**]**.helpers({

options**:** **function**() {

**return** Options.find({testId**:** **this**.testId, questionId**:** **this**.\_id}, {sort**:** {rank**:** 1}});

},

isRadio**:** **function**() {

*// option context (inside {{#each options}}{{/each}})*

**return** Template.parentData(1).type **===** 'radio';

},

hasTimer**:** **function**() {

**return** **!!this**.timer;

}

});

Template**[**getTemplate('editable\_question')**]**.rendered **=** **function**() {

**var** self **=** **this**;

**if**(**this**.data.timer **>** 0) {

$(**this**.firstNode).find(".timer").knob({

min**:** 0,

max**:** 100,

width**:** 100,

height**:** 100,

'release' **:** **function** (val) {

check(val, Number);

**if** (val **<** 0) {

**throw** **new** Meteor.Error('Timer can\'t be negative number');

}

Questions.update({\_id**:** self.data.\_id}, {$set**:** {timer**:** val}});

}

});

$(**this**.firstNode).find(".timer").val(**this**.data.timer);

}

**var** options **=** document.getElementById('option-' **+** **this**.data.\_id);

Sortable.create(options, {

group**:** 'question-' **+** **this**.data.\_id,

sort**:** **true**,

draggable**:** ".option-wrapper",

handle**:** '.option-handler',

ghostClass**:** "option-ghost",

animation**:** 150,

onEnd**:** **function** (evt) {

**var** before, after, newRank;

before **=** evt.item.previousElementSibling;

after **=** evt.item.nextElementSibling;

**if** (**!**before) {

*//if it was dragged into the first position grab the*

*// next element's data context and subtract one from the rank*

newRank **=** Blaze.getData(after).rank **-** 1;

} **else** **if** (**!**after) {

*//if it was dragged into the last position grab the*

*// previous element's data context and add one to the rank*

newRank **=** Blaze.getData(before).rank **+** 1;

} **else** {

*//else take the average of the two ranks of the previous*

*// and next elements*

newRank **=** (Blaze.getData(after).rank **+**

Blaze.getData(before).rank)**/**2;

}

*//update the dragged Question's rank*

Options.update({\_id**:** Blaze.getData(evt.item).\_id}, {$set**:** {rank**:** newRank}});

},

});

};

Додаток В

Лістинг «editable\_question.js»

<template name="editable\_question">

<section class="question-wrapper col-xs-12 col-md-12">

<div class="question-card list-group-item well col-xs-12 col-md-12">

<li class="question-text" contenteditable>**{{**question**}}**</li>

<div id="option-**{{**\_id**}}**" class="question-options">

**{{**#each options**}}**

<div class="option-wrapper">

<i class="option-handler mdi-navigation-more-vert pointer"></i>

**{{**#**if** isRadio**}}**

<div class="option-item radio">

<label>

<input type="radio" name="question-**{{**..**/**\_id**}}**" value="**{{**string**}}**" class="radio-option"><span class="circle"></span><span class="check"></span>

</label>

<span class="option-text" contenteditable>**{{**string**}}**</span>

</div>

**{{else}}**

<div class="option-item checkbox">

<label>

<input type="checkbox" name="question-**{{**..**/**\_id**}}**" value="**{{**string**}}**" class="checkbox-option"><span class="ripple"></span><span class="check"></span>

</label>

<span class="option-text" contenteditable>**{{**string**}}**</span>

</div>

**{{/if}}**

</div>

**{{/**each**}}**

</div>

<div class="add-option-btn pointer"><h6>+ Add an option</h6></div>

<footer>

<div class="enable-checkbox-btn pointer"><h6>Multiple</h6></div>

<span>|</span>

<div class="enable-radio-btn pointer"><h6>Single</h6></div>

<span>|</span>

<div class="enable-timer-btn pointer"><h6>Add timer</h6></div>

<span>|</span>

<div class="remove-question-btn pointer text-danger"><h6>Remove</h6></div>

</footer>

<button class="btn btn-fab btn-fab-mini btn-raised btn-sm btn-unapproved approve-btn">

<i class="mdi-navigation-check"></i>

</button>

**{{**#**if** hasTimer**}}**

<div class="timer-wrapper">

<input type="text" class="timer">

</div>

**{{/if}}**

</div>

</section>

</template>