UDC 004.418

Kudrenko S.O.,
orcid.org/0000-0002-0759-3908,
Stoliar A.L.

DATA ANALYSIS SYSTEM LOGICAL DESIGN AND IMPLEMENTATION

National Aviation University

stanislava@i.ua
stoliarannanau@gmail.com

Introduction

Automation has the most impact on information, mainly due to the need for quick search, processing, selection, and access to it. In this paper, we consider a design and implementation data analysis system, more precisely, an information system for automating sociological research processes, namely questionnaires. In work [3] the requirements had been assessed and the required technology stack selected. Following the definition of the technology stack, proceeded to the allocation of tasks and subtasks, determining the estimated time to complete each task.

Problem statement

The design of single information storage is one of the goals sought in data analysis automation. When developing automated systems, having a single data storage center is also critical to reduce individual user functions.

Another necessity for the system under development is the construction of a user interface that is simple and free of excessive information and functionality.

The most successful technique or combination of methods that would produce the intended outcome was chosen to provide the best testing quality for each method and component of a tested program.

User Interface Design

The web application interface is the visual component of the program with which the user interacts while using it. The user interface (UI) should be straightforward, intuitive, welcoming, consistent, and aesthetically appealing. The color scheme and overall concept under which the application is created determine the visual components. A vibrant and cheerful design, following the concept of space and adventure over unlimited distances, was chosen to fulfill the objective of a good perception of the web application's interface.

The user experience (UX) is determined by the proper layout of interface components, logical sequences of web application situations, and obvious outcomes of user interaction with the interface.

The following components are included in the application:

- pages for logging in and registering;
- the dashboard;
- the user's personal information page;
- a questionnaire page;
- a questionnaire creating form;
- a page with a viewer’s summary.

A user's email address and password are required to log in. The user is forwarded to the dashboard page after logging in. All general analytical information is available on this page. (fig.1). The user may view the total number of questions that have been answered across all questionnaires.

The percentage of the questionnaire that has been completed is calculated on the tab with the finished questionnaire. Some questionnaires may be completed instantly, while others can be stopped. The percentage is derived using the correlation concept, which compares the total number of initiated questionnaires to those that are wholly finished.

There is also the option to produce a report for each questionnaire. When numerous participants complete a questionnaire, the system user can generate a report using the questionnaire's analytical data.

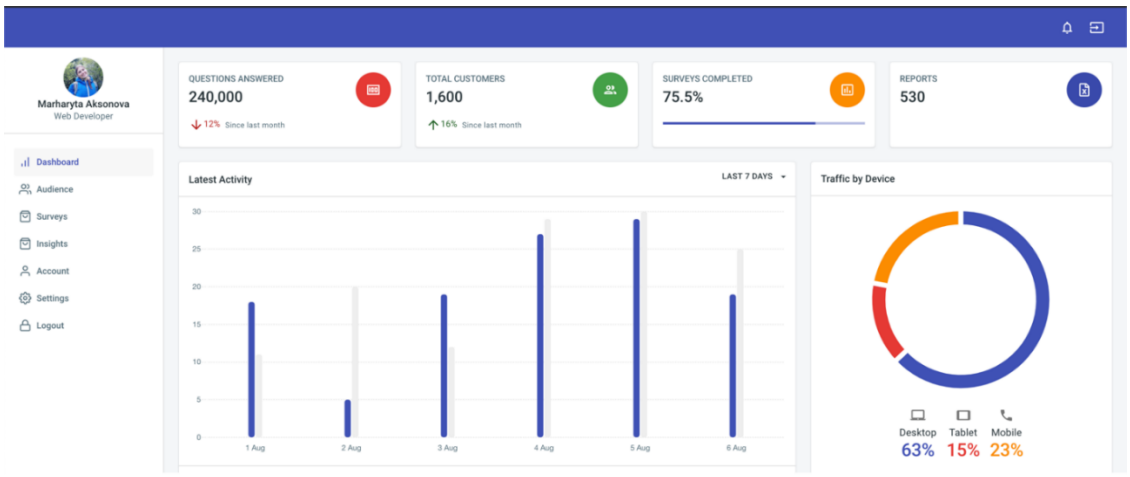

Fig. 1. Dashboard Page Screenshot

Two charts may be found on the dashboard page. The graph depicting the most recent activity, which shows how actively participants finished the questionnaires. The grey bar displays how many questionnaires have been started by participants by the respective date.

The number of questionnaires completed is indicated by the blue bar. If the grey bar is greater than the blue bar, it indicates that some of the questionnaires that have been started have not been finished. If the blue bar is greater than the grey bar, it indicates that some participants completed the questionnaire a second time. When the grey and blue bars are at the same height, the best results are obtained.

A react-responsive package checks the screen resolution to determine the user's current device and render the fitting UI.

The graph on the right shows which device consumers used to respond to the questionnaire's questions. To complete the questionnaires, one of three devices: a PC, a tablet, or a mobile phone can be used.

Just two buttons can be seen on the taskbar. If a notice is received, the notification indicator will appear. There is currently no rationale for notifications. It is expected to expand after the system's fundamental functionality has been developed and tested. The user can log out of the system by clicking the second button. A popup containing the button explanation appears when the user hovers over these buttons.

Server and Database Design

Express.js is used to create the server. Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications [2]. At the root of the program, a server file is produced, which starts the server in the static subdirectory (fig.2).

The environment variable or code can be used to identify the port on which the server will be built.

A web application, regardless of how sophisticated, cannot function without data and, as an outcome, a database. It is vital to identify what will be stored in the database and what the service will require before selecting a database.

The objective is to acquire a broad and full understanding of the database's architecture.

There is no requirement for a relational database in this project, thus an object database was chosen instead.

For this job, MongoDB is an effective and scalable database. MongoDB is a document database designed for ease of development and scaling [4]. MongoDB is a result of the symbiotic relationship between relational databases and key-value storage that has proven to be quite effective.

All data are organized into collections, and global connections between collections have already been built.

The questionnaire analysis system provides the job of writing the responses of consumers to a database and allowing them to be reviewed. Questions and other data from questionnaires are saved.

To complete this work, a form and a function that is invoked once the form is confirmed should be built.

When uploading data to the cloud, the initial step is to determine which document will be used to store the data.

It's possible to achieve this with MongoDB by utilizing a document reference and the path to the needed document.

Collections and documents must be used to identify the route. Collections should come first, followed by documents. Figure 3 shows a snippet of code associated to receiving questionnaires.

```
const express = require('express');
const { graphqlHTTP } = require('express-graphql');

const app = express();

app.use(
  '/graphql',
  graphqlHTTP({
    schema: SystemGraphQLSchema,
    graphiql: true,
  }),
);

app.listen(4000);
```

Fig. 2. Server Code Screenshot

```
import {MongoClient, ObjectId} from 'mongodb'
const MONGO_URL = 'mongodb://localhost:27017/blog'
const db = await MongoClient.connect(MONGO_URL)
const Surveys = db.collection('surveys')
const Users = db.collection('users')


Query: {
  post: async (root, {_id}) => {
    return prepare(await Surveys.findOne(ObjectId(_id)))
  },
  posts: async () => {
    return (await Surveys.find({}).toArray()).map(prepare)
  },
  comment: async (root, {_id}) => {
    return prepare(await Users.findOne(ObjectId(_id)))
  },
},
```

Fig. 3. Receiving Questionnaires Code Screenshot

Testing Strategy

For ensuring the maximum quality level of testing for every method and component of an evaluated application, the most effective technique or combination of techniques should be employed. In this scenario, the best relation between the amount of time spent testing and the quality of testing is required. A collection of tests should be as loss-free as possible while still covering as much of the system's functionality as possible.

Several distinct testing methodologies will be employed during the app's testing. These approaches may be classified into two groups: white-box methods and black-box methods. The distinction among the two types of methodologies is that black-box testing takes place without access to the app's source code. It simply implies that the tester has access to the same options as the app's user. The tester utilizes the software code to obtain the desired effect when employing white-box approaches. For linear structural functions,

black-box testing may be used. Black-box tests use the strategies of comparable partitions and error assumptions to verify the specifications [1].

The white-box technique should be used to test methods of classes with a nonlinear structure. When testing highly sophisticated procedures, it's also required to use the black-box technique, which involves placing the error hypothesis [1].

This tactic will improve the system's and procedures' dependability, which is extremely vital for the job. This strategy is ideal for evaluating the app's interface.

After all components and methods have been tested, the most crucial step is to test the entire application. This step shouldn't be overlooked since certain issues may not show up when testing individual components separately.

So, for testing the developed system the following strategy was developed:

- Choosing error and conditions assumption approaches;
- Testing will be done bottom-to-top (submodules are tested first);
- Testing key functionalities and components using the chosen methods;
- Testing other functionalities and components using the error assumption technique.

It's also crucial to provide unit testing to ensure that the functionality is operating properly.

Unit testing with snapshots enables the creation of relatively basic and easy-to-maintain tests, which will aid in the prevention of issues in the development and improve the application's reliability (fig.4).

A snapshot is produced for each test case and is used as a reference (fig.5).

Following the application development, a test was added to ensure that the components render appropriately. React-testing and jest libraries were used to shallow the components. Only the components that are being tested have to be rendered, hence shallow rendering is required.

The assert mechanism is used to build a snapshot for the component. It compares the rendering result of the component to the reference snapshot. If no reference exists, one will be established when the test is performed for the first time.

A snapshots folder appears in the component's folder after the test start, where all captured snapshots would be stored. The snapshots need to go into the git repository. Snapshots are supplementary graphic records that demonstrate the way the project evolved throughout the development.

Result Evaluation

React creates the marking for each component and monitors events that are part of the User Timing API while in development mode [5].

To construct a performance profile, the programmer needs to navigate to the localhost and use the developer tools: click the Start profiling and refresh the page button.

This action will begin the recording of data regarding the current page's performance. After the webpage is fully loaded and displayed to the user, the browser will immediately stop collecting data. The red bar indicates the presence of a significant CPU load at the point in the timeframe. This may be the cause of the app's slowness, and it's something to be investigated further.

Different activities are represented by the colors in the graphic at the top edge of the screen (fig.6). The decrease in efficiency that occurs throughout diverse sorts of operations has its own set of factors. There are certain methods for resolving and assessing diverse issues.

The User Timing API, which could be utilized to add time markers in different applications, publishes metrics for React [6]. It enables professionals to see the amount of time required for a component to load or how long it'd take for an event to occur. The browsers do not have incorporated React debugging tools, although some do include support for the React API [5]. Furthermore, while all modern browsers support the User Timing API, the Chrome developer tools' Performance tab went far beyond competitors, making troubleshooting React apps in Chrome far simpler.

```
  it('calls onClick props when clicked', () => {
    const onClick = jest.fn();
    render(<Button onClick={onClick}>Click Me</Button>);
    fireEvent.click(screen.getByTestId('atoms-button'));

    expect(onClick).toHaveBeenCalledTimes(1);
  });

  it('is disabled when disabled prop equals true', () => {
    render(<Button disabled>Click Me</Button>);
    const button = screen.getByTestId('atoms-button');

    expect(button).toBeDisabled();
  });

  it('has properly className', () => {
    render(<Button className="secondary">Click Me</Button>);
    const button = screen.getByTestId('atoms-button');

    expect(button).toHaveClass('govuk-button--secondary');
  });

  it('has properly text content', () => {
    const text = 'Click Me';
    render(<Button>{text}</Button>);
    const button = screen.getByTestId('atoms-button');

    expect(button).toHaveTextContent(text);
  });
});
```

Fig. 4. Button Unit Test Code Screenshot

```
// Global imports
import React from 'react';
import { render, fireEvent, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';

// Local imports
import Button from '../../Components/Atoms/Button';

describe('<Button />', () => {
  it('renders and match the snapshot', () => {
    const onClick = jest.fn();
    const { asFragment } = render(
      <Button buttonVariant="secondary" disabled={false} onClick={onClick}>
        Test
      </Button>
    );

    expect(asFragment).toMatchSnapshot();
  });
```

Fig. 5. Button Snapshot Test Code Screenshot

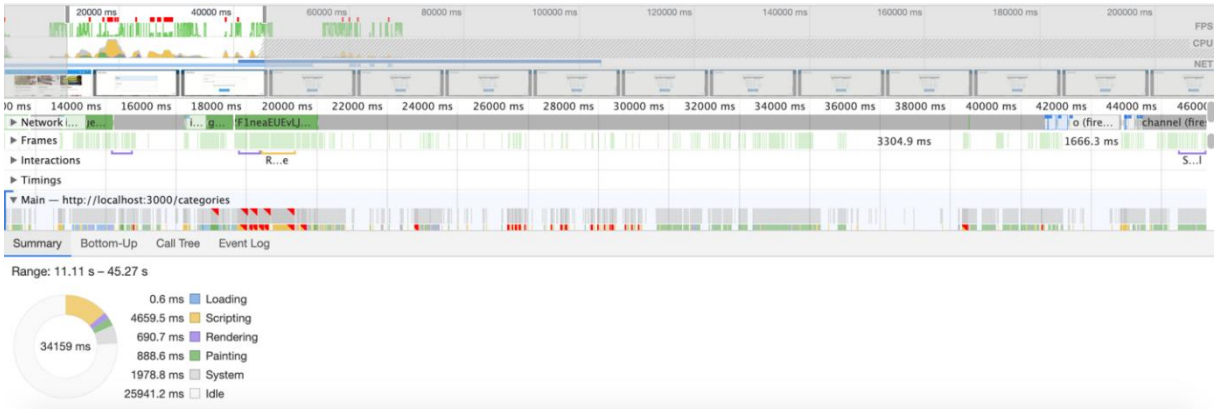

Fig. 6. System Performance Results Screenshot

User Manual Development

To use the application, the user must first signup. They may do it by inputting their information: email, password, and personal details. The user will gain access to all system

features once the registration process is complete. These features include the following:

Adding a questionnaire. To create a new questionnaire, go over to the section containing a list of questionnaires and click the 'Add' button, or go straight to the page with the form. Afterward, the user needs to create a form in which they would input the questionnaire's title, possibly its description, and a series of questions. The questionnaire will be included in a database after form confirmation.

Editing a questionnaire. To modify a questionnaire, the user must first go to the questionnaire list, choose the questionnaire they want to update, and click the edit button. Afterward, the user is sent to a webpage with a form where they may amend and conclusively confirm the information. The outdated data will then be replaced by new information in the database.

Deleting a questionnaire. To delete a questionnaire, the user needs to go to the page with a list of questionnaires, choose one that they want to remove, and click the delete button. For the user, a verification popup is displayed. If the user chooses to confirm, the questionnaire will be erased from the database; otherwise, nothing will change. There's also the option of deleting many questionnaires at once. Simply pick a few questionnaires and click the delete button to remove them.

Searching capabilities. To find a certain questionnaire, the user needs to type the request into the system's search box. Once the search is complete, the user will see the found objects. If no entries are discovered, the appropriate notification will appear.

Further Development Direction

Any system, including this one, has a path to follow in order to grow and evolve. The key features and capabilities were implemented to make the system work as an MVP during this phase of development.

There are a bunch of features that could be incorporated to the existing system. It would be a useful option to include notifications. Because a huge database of questions may be incorporated within the framework of this app and adapted into any questionnaire style, the application's potential is fairly broad.

Conclusions

Following the defining of the technological stack, tasks and subtasks were assigned, and the expected time to perform each task was calculated.

The UX was determined by the proper layout of interface components, logical sequences of web application situations, and obvious outcomes of user interaction with the interface. UI followed a bright and exciting design concept.

The following components are included in the application:

- pages for logging in and registering;
- the dashboard;
- the user's personal information page;
- a questionnaire page;
- a questionnaire creating form;
- a page dedicated to questionnaire analysis;
- a page with a viewer's summary.

It was decided what to keep in the database and what the site would require before selecting a database. The objective was to gain a broad and thorough picture of the database's structure.

The course of testing needed to check the app's proper functionality was decided on. It was chosen to use white-box and black-box methods of testing, that were described.

Basic functionality of the app was implemented and the plan for future expansion decided.

References

1. Differences between Black Box Testing vs White Box Testing. – [Electronic resource]. Access mode: https://www.geeksforgeeks.org/differences-between-black-box-testing-vs-white-box-testing

2. Fast, unopinionated, minimalist web framework for Node.js. – [Electronic resource]. Access mode: https://expressjs.com

3. *Kudrenko S.A.* Method for complex objects automated design on autodesk revit based / Kudrenko S.A., Fomina N.B., Kramarenko I.P. / Проблеми інформатизації та управління. – 2021. – V. 65. – P. 64-74.

4. MongoDB. – [Electronic resource]. Access mode: https://www.mongodb.com

5. React. A JavaScript library for building user interfaces. – [Electronic resource]. Access mode: https://reactjs.org/

6. User Timing API. – [Electronic resource]. Access mode: https://developer.mozilla.org/en-US/docs/Web/API/User_Timing_API

Kudrenko S.O., Stoliar A.L.

DATA ANALYSIS SYSTEM LOGICAL DESIGN AND IMPLEMENTATION

The main advantage of the automation process is that it allows to reduce the amount of required memory, reduce the time for data processing, and reduce the number of copies of documents when updating information.

The choice of technologies for developing an application is an important stage which has been described in paper. Before developing the system, the requirements should be carefully prepared and described. A well-chosen combination of technologies should ensure comfortable work in the future at all stages of the application's existence

Obviously, the technology stack should be easily scalable, functional, correspond the latest market trends. Most importantly, it has to be easily supported in the future by other developers.

React.js has a capacious and understandable API. To work with React, it is necessary to understand a number of terms and the differences between them. Its popularity continues to grow and it is at the heart of many projects.

Keywords: *automation data analysis, Application Programming Interface, React.js, MongoDB.*

Кудренко С.О., Столяр А.Л.

ЛОГІЧНЕ ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ АНАЛІЗУ ДАНИХ

Головною перевагою процесу автоматизації є те, що він дозволяє зменшити обсяг необхідної пам'яті, скоротити час на обробку даних та зменшити кількість копій документів при оновленні інформації.

Вибір технологій для розробки додатків є важливим етапом, який був описаний у роботі. Перш ніж розробляти систему аналізу даних, слід ретельно підготувати та описати вимоги. Правильно підібрана комбінація технологій повинна забезпечити комфортну роботу в майбутньому на всіх етапах існування програми

Очевидно, що стек технологій повинен бути легко масштабованим, функціональним, відповідати останнім тенденціям ринку. Найголовніше, що в майбутньому його повинні легко підтримувати інші розробники.

React.js має місткий і зрозумілий API. Для роботи з React необхідно розуміти ряд термінів та відмінності між ними. Його популярність продовжує зростати, і це в основі багатьох проектів.

Ключові слова: *автоматизація аналізу даних, прикладний програмний інтерфейс, React.js, MongoDB.*