The process of developing a web-based system on the basis of cinema website models *

Lukina Daria Vasilevna
Kazan Federal University
Kazan, Russia
dasha.luckina2015@yandex.ru

Medvedeva Olga Anatolievna
Kazan Federal University
Kazan, Russia
OAMedvedeva@kpfu.ru

Mustafina Svetlana Anatol'evna
Bashkir State University
Ufa, Russia
mustafina_sa@mail.ru

Abstract

This article discusses the steps involved in developing a website based on existing models. The subject area of sites with similar topics is analyzed. The selection of the necessary tools and technologies for development has been identified and justified. The site development process is described, as well as the result of testing the program.

Introduction

One of the most important types of web-based systems is the site. Sites contain a large amount of important and necessary information. They are available for any user, therefore they are a relevant and sought-after service at the present time. Any website is designed for a specific group of users, united by common interests and requirements, who are looking for information on a specific topic. If the user can find all the information he needs in one place where all aspects of the question posed by the visitor are presented in detail, such a web-system will be very much appreciated. The site for booking and selling tickets at the cinema will not only allow users to purchase tickets in a matter of seconds, but also book online. Also, the client will be able to view the latest relevant information of the cinema. Visitors will be attracted to such sites that have a beautiful design, are not burdened with excessive graphics, animation, advertising and just will be interesting to users. Also affects the speed of page display and the correct display in the browser window, so it is important that the site meets all the requirements of the visitor.

The aim of this project is the acquisition and use of theoretical information on the design of a website and the application of practical skills in its development.

To complete the work, the following tasks were set:

1. analysis of a certain number of sites with similar topics to identify the user needs in demand;
2. organize the structure of data storage;
3. develop the server side of the site;
4. develop a client part.

1. Domain Analysis

During the analysis of the subject area, the main operations were identified: viewing existing films and their schedules, selling and booking tickets, registration and authorization. Based on these requirements, the roles were defined: administrator and user, and the following use cases:

1. Log in.
2. Register in the system
3. Enter movie data.
4. Enter session data.
5. Select a session.
6. Buy or book a ticket.

2. Designing the information storage system prototype

From the use cases, we can conclude which tables are necessary for the implementation of the site. To correctly compile the database, you must first present it in the form of an ER model. The ER model shown in Figure 1 was created in Draw.io; the database itself is stored in SQLite.

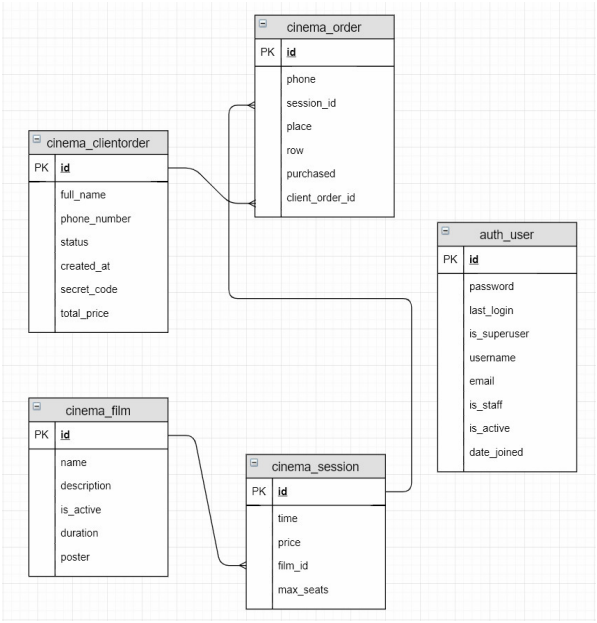

Figure 1: ER database model

3. Development of the prototype information system interface

During the design of the interface, a prototype was developed. The main colors are: raspberry, burgundy, gray. The background is a dark full-screen image that simulates a cinema hall. This image is static (i.e., it does not move when scrolling). Elements requiring user attention are painted in crimson, Burgundy was taken to highlight some elements that merge with the background. Designing the interface was done using Photoshop.

4. Development of a software module for an information system

Project development consists of two parts: server and client. For each of them, different software tools were selected. For the server part of the project, the programming language “Python” and the web-framework Django were chosen. Development will take place in the PyCharm IDE. For the client part, the JavaScript language was chosen as the programming language, the library for developing the interface is React.js.

4.1. Server side development

To work with data, django-orm is used. The description of the database structure is as follows: first, the tables in the database are described (Figure 2).

```python
from django.db import models


class ClientOrder(models.Model):
    """Client order model"""

    STATUS_CHOICE_BOOK = 0
    STATUS_CHOICE_BUY = 1
    STATUS_CHOICES = (
        (STATUS_CHOICE_BOOK, 'Preserve'),
        (STATUS_CHOICE_BUY, 'Bought'),
    )

    full_name = models.CharField(verbose_name='FIO', max_length=128)
    phone_number = models.CharField(verbose_name='Phone number', max_length=10)
    status = models.CharField(verbose_name='Status', choices=STATUS_CHOICES, max_length=20)
    created_at = models.DateTimeField(verbose_name='Created', auto_now_add=True)
    total_price = models.DecimalField(verbose_name='Sum of order', max_digits=6, decimal_places=2, null=True)
    secret_code = models.CharField(verbose_name='', max_length=50, null=True)

    def __str__(self):
        return self.full_name

    class Meta:
        """Meta"""

        verbose_name = 'Order'
        verbose_name_plural = 'Orders'
```

Figure 2: description of the database structure

The project's root directory (Figure 3) contains the following files: apps (a folder containing framework applications), a conf directory containing basic routing, a file for deploying the application on the destination server, and also a settings file where you can change the language, time zone, database and much more.

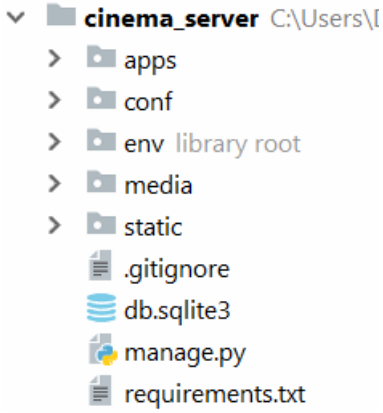

Figure 3: root directory

Env is a virtual environment containing a separate Python interpreter with its packages related only to this project; a list of dependencies can be found in the requirements.txt file. The framework defines the basic structure of the site, a standard version of the application may look like this:

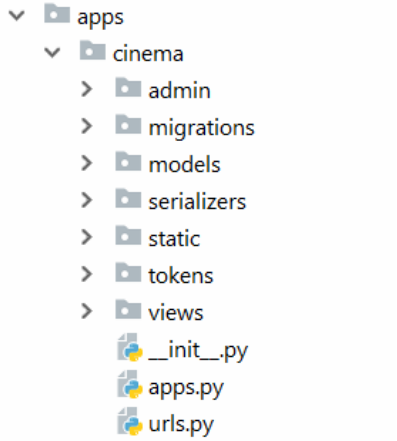

Figure 4: application organization structure

Static and media, the directory where the statics and media files are located, respectively, scripts, styles and more belong to statics, media files: videos, pictures, etc.
Admin - a directory containing settings related to the admin panel, for example, how many entries to display on the page or which fields to display. Database models are also registered here so that they can be accessed through the administrator interface.
Migrations - the directory in which the automatically generated migration files are stored when the database structure is changed.
The Model folder contains the database models, the description of tables on django orm.
The Serializers directory contains classes for serializing complex objects in Json, based on classes from the django-rest-framework.
The view directory contains the logic for generating a response for different types of queries, accessing the database and classes for serialization. The data from the POST request is taken, and a new record is created in the database.
Requests come from the routes represented.
In this part, the path to the views contained in the views directory is mapped.

4.2. Client side development

The client part is written using JavaScript ES6 and the React.js library. The main structure is created using the create-react-app utility; it creates the basic template and server for development (Figure 5).
The main actions take place in the src directory. Components - components that are transferred to the user. Utils are small utilities. For example, a utility that sets headers for requests when there is an authorization token.

The main file is the index.js file.

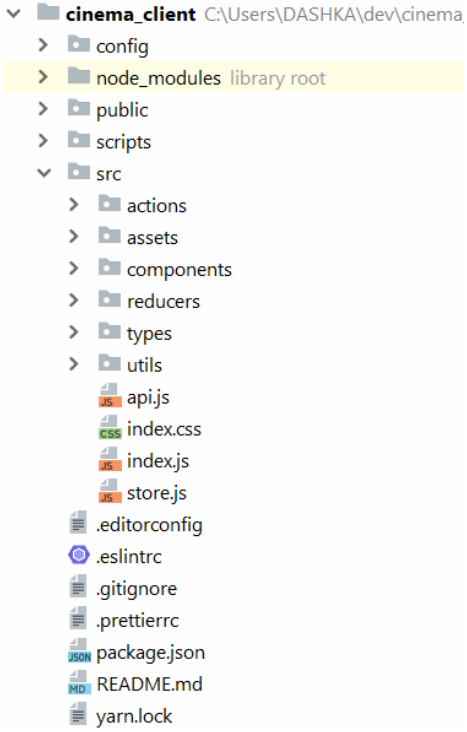

Figure 5: client file structure

In this file, the project adds the router and the main repository from the redux libraries, react-router. A check is underway for user authorization and whether the JWT token is still valid.
In this file, the project adds the router and the main repository from the redux libraries, react-router. A check is underway for user authorization and whether the JWT token is still valid. After verification, the App component is mounted on the html page. The App component is implemented as follows: it places the so-called navbar and footer on the page, leaving space between them to render the components that depend on the route. Thus, when switching, for example, to the address 127.0.0.1haps000/login, the LoginPage component will be mounted. The Api file describes how to access the server using the axios library. Due to this, the server ip is stored in one place, and when changing endpoints they can be fixed in one file, and instead of requests like: http://127.0.0.1:8000/api/cinema/films/5 you can write it like this: api. film.fetchOne (5). An example of a functional component, which if the user is not logged in, draws a page with the registration form for him, otherwise redirects to the main page.

```
const SignupPage = ({ isAuthenticated, signup }) =>
  !isAuthenticated ? (
    <div className="container">
      <div className="col-md-4 offset-md-4 col-sm-2 col-sm-8">
        <div className="sign-up">
        <SignupForm onSubmit={signup} />
          </div>
      </div>
    </div>
  ) : (
    <Redirect to="/" />
  );
```

Figure 6: functional component example

Reducer example from redux library (Figure 7). The file describes how and what information will be stored in the global storage, from where any component can take information. The reducer does not change the storage data, but creates a new one, based on old data and new ones.

```
const initialState = {
  token: '',
  username: '',
};

const userReducer = (state = initialState, action = {}) => {
  switch (action.type) {
    case userTypes.USER_LOGIN:
      return {
        ...action.payload,
      };
    case userTypes.USER_LOGOUT:
      return {};
    default:
      return state;
  }
};

export default userReducer;
```

Figure 7: reducer example

Actions are needed to interact with the repository. FilmAddedOne is an action to add a new movie (Figure 8). FilmFetchOne is an asynchronous action that can cause several common ones, in which case a request to the server is made in it, and after that the data is stored in the storage (Figure 9).

```
const filmAddedOne = film => ({
  type: filmTypes.FILM_FETCH_ONE,
  payload: film,
});
```

Figure 8: action example

```
export const filmFetchOne = id => dispatch =>
  api.film
    .fetchOne(id)
    .then(res => dispatch(filmAddedOne(res)))
    .catch(err => err.message);
```

Figure 9: asynchronous action

5. Testing the prototype information system for booking and selling tickets at the cinema

During testing, the method of 50 test cases was applied, which checks the operation of the site. Its essence is that it checks the operation of the main functions of the site and its correct display in all current browsers. During testing, the method of 50 test cases was applied, which checks the operation of the site. Its essence is that it checks the operation of the main functions of the site and its correct display in all current browsers.

6. Conclusion

Modern cinemas give people the opportunity to buy and book tickets online without taking up huge queues. The developed site allows users to use this feature. As a result, it should be noted that during the development all the tasks were solved:

1. analysis of a certain number of sites with similar topics to identify the user needs in demand;
2. organize the structure of data storage, learn how to develop the server side of the site
3. develop the server side;
4. develop a client part.

In the first part of the article, an analysis of the subject area was carried out and the requirements for the system were identified. The possibility of creating a site using the client-server system is considered. The second describes the data storage structure. In the third part - designing the user interface. In the fourth part - the process of developing the client part, in the fifth - the process of developing the server part. The developed site meets all the requirements set at the stage of setting the task. When developing the web site, ready-made modules were used,

which were modified taking into account the specifics of the web site and were successfully implemented in its structure.

Acknowledgements

The authors express their gratitude and appreciation to the organizers of the conference of the Ural Federal University and Innopolis University for their help and support in preparing the article.

References

1. Andrianova A.A., Mukhtarova T.M., Rubtsova R.G. Laboratory workshop on the course 'Database Technologies': Textbook / A.A. Andrianova, T.M. Mukhtarova, R.G. Rubtsova. - Kazan: KFU, 2016 .- 97 p.
2. Architecture and design of software systems: monograph / S.V. Nazarov. - 2nd ed., Revised. and add. - M.: INFRA-M, 2018 .- 374 p.
3. Boilerplate with a pre-configured server for development and much more. [Electronic resource] - Access mode: URL: https://reactjs.org/docs/create-a-new-react-app.html.
4. Documentation for the react.js library [Electronic resource] - Access mode: URL: https://reactjs.org/
5. Documentation on bootstrap» [Electronic resource] - Access mode: URL: https://bootstrap-4.ru/
6. Introduction to software engineering: Textbook / V.A. Antipov, A.A. Bubnov, A.N. Pylkin, V.K. Stolchnev. - M.: COURSE: INFRA-M, 2017 .- 336 p.
7. Js library for global storage in the application. [Electronic resource] - Access mode: URL: https://redux.js.org/
8. Js library for reactjs for routing. [Electronic resource] - Access mode: URL: https://reacttraining.com/react-router/
9. Library for convenient CSS in Js. [Electronic resource] - Access mode: URL: https://www.styled-components.com/
10. Module for creating api. [Electronic resource] - Access mode: URL: https://www.django-rest-framework.org/
11. Node.js. [Electronic resource] - Access mode: URL: https://nodejs.org/en/ (02.20.2019)
12. Python web-framework. [Electronic resource] - Access mode: URL: https://www.djangoproject.com/