Создание интерфейса для умных объектов на примере бассейна

А.Г. Клименко, К.С. Зайцев

***Аннотация*. Разработка умных объектов включает в себя решение широкого круга связанных между собой задач. Одной из таких задач является разработка пользовательского интерфейса. Целью настоящей работы является описание разработанного подхода к процессу разработки интерфейса умного объекта на примере системы «Умный бассейн». Определяются основные принципы разработки пользовательских интерфейсов, продумывается сценарий использования веб-приложения, строятся диаграммы use-case и user-flow. Затем выбираются технологии разработки с учетом современных тенденций frontend-разработки и настраивается рабочее пространство. Результаты выбора технологического стека, полученные авторами, оформлены в виде таблицы средств реализации интерфейсов. После этого описываются особенности работы с библиотекой React, такие как возможность создания собственных или использования встроенных компонентов, возможность настройки динамической маршрутизации, возможность использования текущего состояния элементов и обработка событий с применением хуков useState и useEffect. Приводятся примеры кода для демонстрации всех перечисленных особенностей.**

***Ключевые слова* – умный объект, интернет вещей, пользовательский интерфейс, веб-приложение, диаграмма use-case, диаграмма user-flow, TypeScript, React**

I. ВВЕДЕНИЕ

В настоящее время растет стремление подключить все к интернету, не только для того, чтобы отправлять информацию на сервер, но и для по-лучения полного контроля над физическими устройствами [1]. Такой рост приводит к появлению различных сложных приложений, которые предполагают взаимодействие устройств друг с другом с помощью интернета.

Однако, разработка приложений является сложной задачей из-за большого разнообразия аппаратных и программных технологий, которые взаимодействуют в системе IoT, а также включает в себя решение широкого круга связанных между собой задач. Одной из таких задач является разработка пользовательского интерфейса, с помощью которого человек может взаимодействовать с системой.

В работе будет рассмотрена разработка клиентской части умного объекта на примере системы «Умный бассейн», основная задача которой заключается в том, чтобы изменять скорость течения воды в бассейне в зависимости от пульса пользователя. Для этой системы предусматривается приложение, с помощью которого оператор системы (врач, тренер, и проч.) сможет контролировать или изменять параметры текущей тренировки. Эта система найдёт применение в реабилитационных центрах, спортивных комплексах и других организациях, где необходимо проводить персонализированные тренировки и отслеживать прогресс конкретного пользователя в условиях ограниченного пространства.

II. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ

Перед началом разработки клиентской части приложения необходимо определить принципы, которых нужно придерживаться при разработке интерфейса – как проще интерпретировать контент, как направлять разработку интерфейса по мере развития технологий и требований, стоит ли пользоваться шаблонами успешных интерфейсов или это наложит ограничения на инновационный подход к разработке.

Затем необходимо продумать сценарий использования приложения, описать взаимодействие системы с пользователем со стороны системы по средствам use-case диаграммы.

Следующий шаг – разработка user-flow диаграммы пользовательского интерфейса, которая позволяет взглянуть на взаимодействие системы с пользователем со стороны пользователя.

Затем нужно выбрать технологии разработки и настроить рабочее пространство.

После этого можно наконец приступать к самой разработке, которая представляет собой создание структуры приложения, размещение компонентов на страницах, настройку динамической

Статья получена 16 мая 2022.
Клименко Анисия Геннадьевна, Национальный Исследовательский Ядерный Университет МИФИ, магистрант, email:Klimenko.an11@gmail.com
Зайцев Константин Сергеевич, Национальный Исследовательский Ядерный Университет МИФИ, профессор, email:KSZajtsev@mephi.ru

маршрутизации и обработку событий, то есть действий, которые пользователь совершает с использованием компонентов.

2.1. Принципы проектирования дизайна пользовательского интерфейса

Человек, пользуясь системой, должен пытаться понять, что происходит во время взаимодействия с интерфейсом, и, поскольку большая часть действий скрыта, он должен полагаться на логические выводы. В зависимости от своего понимания работы системы или выполняемой задачи пользователь может делать правильные и неправильные выводы, а, следовательно, у него будет складываться собственное представление о технологии, о системе в целом и о том, как она работает. Такие представления могут быть схожими у многих пользователей.

Авторы статьи [2] считают, что взаимодействие человека с системой можно рассматривать как когнитивный процесс, причем считается, что человек обладает ментальными моделями – личным, часто своеобразным взглядом на то, что и как делает система. С помощью этих моделей пользователь целенаправленно изучает и использует информацию.

Источником ментальной модели является предшествующий опыт [3], особенно с похожими или сопутствующими продуктами. Если пользователь работал с другой, похожей системой или более старой версией существующей системы, то этот опыт обязательно повлияет на восприятие новой технологии, так как пользователь уже обладает сложившимися ментальными моделями.

Для разрабатываемой системы «Умный бассейн» может быть актуален только вариант аналогичной системы. После изучения структуры похожих интерфейсов [4, 5] можно сделать вывод, что стоит разделять страницы настройки профиля пользователя и основного экрана запуска процесса тренировки. Такой шаблон встречается во многих интерфейсах. Как показано в книге [6], «шаблоны часто помогают направить энергию на решение новых проблем, а не на изобретение велосипеда».

Некоторые теоретические взгляды на когнитивную архитектуру подчеркивают ограничения памяти и внимания людей. Приемлемой считается последовательность взаимодействия, которая разработана таким образом, чтобы свести к минимуму нагрузку на кратковременную память пользователя. Например, не нужно предлагать пользователю выбирать из чрезмерного количества пунктов меню или требовать, чтобы он запоминал символы или числа при переходе от одной страницы к другой. Необходимо разрабатывать интерфейс так, чтобы вся нужная информация в интерфейсе предоставлялась пользователю по мере необходимости.

Таким образом, стоит придерживаться шаблонного подхода к разработке, так как система будет проще масштабироваться, когда можно оперировать готовыми модулями и компонентами. Шаблонные интерфейсы выглядят привычнее для пользователя, а значит, он сможет быстрее и эффективнее решать свои задачи. Также стоит разделять процесс запуска процесса тренировки и выбор спортсмена и режима тренировки, чтобы не перегружать интерфейс.

2.2. Разработка дизайна пользовательского интерфейса

Для того, чтобы разработать сценарий использования интерфейса, необходимо определить, кто является пользователем системы, какова его цель и какие шаги ему нужно предпринять для достижения этой цели.

Пользователем системы является оператор, то есть врач, тренер и т.д., – человек, который запускает и завершает процесс тренировки для спортсмена. Также роль оператора заключается в том, чтобы контролировать процесс тренировки и в случае критических ситуаций увеличивать или уменьшать скорость течения воды в бассейне. Для этого ему нужно пройти процедуру авторизации (или регистрации, если оператор еще не пользовался системой), затем выбрать конкретного спортсмена из списка или создать нового, подобрать подходящую для него тренировку из списка или создать новую, наконец, приступить к самой тренировке. Во время тренировки оператор может останавливать и запускать процесс тренировки, изменять скорость противотока в ручном режиме, устанавливать оптимальные значения температур воды в бассейне и воздуха в помещении.

Структурируем и развернем обоснованное выше, и получим диаграмму use-case, показанную на рис. 1.

Диаграмма сценариев использования приложения описывает поведение системы при взаимодействии с оператором, показывает, какой функционал разрабатываемой программной системы будет доступен оператору [7]. Можно выделить 5 базовых вариантов использования – авторизация, регистрация, просмотр списка спортсменов, просмотр списка тренировок и просмотр аналитики процесса тренировки. Некоторые варианты использования имеют включения и расширения.

Рис. 1. Диаграмма сценариев использования приложения.

Диаграмма вариантов использования приложения легла в основу диаграммы user-flow, которая является визуальным представлением последовательности действий пользователя [8, 9]. На рис. 2 представлена диаграмма user-flow, разработанная в онлайн-сервисе Figma.

Рис. 2. Диаграмма user-flow.

Диаграмма user-flow содержит основные эскизы, отвечающие базовым вариантам использования приложения, и дополнительные, которые содержат расширения и включения диаграммы use-case. Полученные эскизы пользовательского интерфейса легли в основу разработанной клиентской части приложения.

III. ВЫБОР ТЕХНОЛОГИЙ РАЗРАБОТКИ

Одним из важных этапов разработки системы является определение технологического стека. Правильно подобранные средства реализации могут сократить время разработки, ее стоимость и повысить качество. Также важно понимать, что выбранные технологии влияют на масштабируемость системы. Поэтому, чтобы избежать проблем в будущем, к выбору технологического стека нужно подходить тщательно.

В наше время ни один даже небольшой проект не обходится без системы контроля версий Git, которая позволяет отслеживать и фиксировать изменения в коде и вести проект одновременно нескольким участникам. Есть несколько популярных сервисов, позволяющих удаленно хранить репозитории. Остановимся на веб-сервисе GitHub, так как он является самым крупным и популярным среди всех перечисленных.

Редактором исходного кода выбран Visual Studio Code, так как он поддерживает множество языков программирования, включает в себя отладчик, подсветку синтаксиса, инструменты для работы с Git, а также содержит большое число расширений и плагинов, упрощающих разработку.

На рис. 3 изображена общепринятая организация логики веб-приложения. Оператор системы взаимодействует только с интерфейсным блоком, который в свою очередь связан только с блоком логики процессов. Блок логики процессов может вызывать блок работы с хранимыми данными. Важно то, что интерфейсный блок не может напрямую отправлять запросы к блоку работы с хранимыми данными, и наоборот. Таким образом, для каждого блока определена своя обязанность, они являются обособленными и работают по принципу «черного ящика».

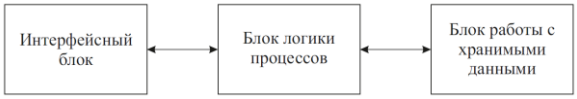

Рис. 3. Организация логики веб-приложения.

В работе рассматривается только интерфейсный блок. Так как блоки обособлены друг от друга, блок логики процессов рассматриваться не будет.

Для создания эскизов пользовательского интерфейса был выбран он-лайн-сервис Figma. Этот редактор обладает рядом преимуществ, в том числе способность работать над одним проектом нескольким людям, кроссплатформенность, возможность создания компонентов и работы с векторной графикой. Для того, чтобы изучить Figma и внедрить ее в работу, достаточно несколько часов. Онлайн-сервис хорошо адаптирован для недизайнеров, обладает высокой производительностью, автоматическим сохранением и системой контроля версий. Также, Figma позволяет создавать библиотеку компонентов – однотипных элементов, которые можно использовать повторно, и которые можно изменять одновременно все сразу, что затруднительно во многих других системах для создания эскизов пользовательского интерфейса.

Клиентская часть – видимая часть веб-приложения, с которой взаимодействуют пользователи. Задача любого разработчика

пользовательского интерфейса – сделать взаимодействие с приложением максимально простым, эффективным и интуитивно понятным. Важно, чтобы пользователь даже при первом взаимодействии с системой понимал, что и как он может сделать.

Изначально для разработки клиентской части были выбраны HTML, JavaScript и CSS. Однако затем, ознакомившись с современными тенденциями frontend-разработки, выбор был сделан в пользу TypeScript и библиотеки React.

Приложение, написанное на TypeScript, удобнее масштабировать, время разработки сокращается благодаря уменьшению числа ошибок в коде, а процесс сопровождения упрощается. При этом TypeScript поддерживается всеми современными фреймворками.

Что касается библиотеки React – ее можно использовать и для простых веб-приложений, и при разработке сложной логики клиентской части, особенно когда в приложении есть большое число повторно используемых компонентов [10]. Библиотека позволяет, например, получать доступ к текущему состоянию элементов и использовать его для обработки событий. Такое поведение эффект называется хуком, и является одним из преимуществ React. Хуки и встроенные компоненты значительно упрощают и ускоряют процесс разработки.

Так как приложение было необходимо разработать в сжатые сроки, в условиях ограниченных ресурсов и без жестких требований к дизайну интерфейса, удобно использовать готовые шаблоны стилевого оформления. Для этого прекрасно подходит библиотека Bootstrap, обладающая минималистичным и интуитивно понятным дизайном.

Для наглядности все выбранные средства реализации собраны в таблице 1.

Таблица 1. Средства реализации интерфейса.

Область	Инструмент	Назначение
Frontend + backend	GitHub	Система управления версиями приложения для совместной работы и единой точки хранения исходного кода
	VS Code	Среда разработки
Design	Figma	Создание эскизов пользовательского интерфейса
Frontend	TypeScript + React	Верстка веб-страниц и реализация бизнес логики приложения
	CSS	Стилевое оформление элементов веб-страниц
	Bootstrap	Единый формат стилевого оформления

IV. РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ

После установки Node.js, плагина ESLint и создания React-приложения, в директории проекта появляется шаблонное дерево файлов. Приложение запускается из автоматически созданного файла src/index.tsx.

Удобство разработки на React заключается в том, что библиотека позволяет создавать компоненты и использовать их или уже готовые повторно, передавая различные параметры в качестве аргументов. Рассмотрим это на примере приветственного экрана, неполный листинг которого представлен ниже.

```
import React from ’ react ’ ;
import { Link }
from ’react-router-dom’;
import ’bootstrap/dist/css/bootstr
ap.min.css ’;
import { Container } from
"react-bootstrap ";
class Splash extends
React.Component { render ()
{ return (
<Container className=’pt-5
container-fluid text-center d-flex
flex-column’>
{/* … */} </Container>
)}}
export default Splash ;
```

В качестве готовых используются компоненты Container из библиотеки react-bootstrap как элемент верстки и компонент Link из библиотеки react-router-dom как элемент маршрутизации. Получаем, что пользовательский интерфейс состоит из независимых, повторно используемых частей, каждая из которых работает независимо друг от друга.

React позволяет определять собственные компоненты как классы или функции [10]. Для этого необходимо расширить класс React.Components и за- тем экспортировать получившийся компонент. Внутри класса обязательно должен присутствовать метод render(), возвращающий верстку экрана, которая может состоять как из уже готовых React-компонентов, так и из html-кода.

Еще одной важной частью веб-приложения является маршрутизация [11]. Она нужна для того, чтобы оператор системы понимал, на какой

странице находится, и имел возможность навигации по истории. Для того, чтобы настроить динамическую маршрутизацию, была подключена библиотека react-router-dom. Ниже представлен неполный листинг компонента App, который задается функцией.

```
import React from ' react ' ;
import { Routes, Route } from 'react-router-dom'; import Splash from './ screens/splash '
// …
function App() { return ( <Routes> <Route path='/' element={<Splash />} /> {/* … */}
</Routes> );}
export default App;
```

Во второй строке из библиотеки react-router-dom импортируется два компонента – Routes и Route. Первый компонент является внешним, а второй – задает маршруты для конкретных элементов. Например, для экрана заставки предусмотрен корневой путь.

Стоит отдельно обратить внимание на еще одну особенность разработки с помощью библиотеки React – хуки. Это нововведение, которое позволяет использовать состояния и другие методы React без создания классов [12]. С помощью хуков можно расширить возможности функциональных компонентов, что упрощает работу и делает код более читаемым.

Хук useState вызывают, чтобы наделить функциональный компонент внутренним состоянием. Вызов useState возвращает текущее значение состояния и функцию для его изменения. Текущее состояние будет храниться между рендерами (процессами опроса актуального состояния компонентов), а функция для изменения этого состояния может использоваться где угодно в коде, причем не раз.

Рассмотрим такое состояние на примере окна просмотра спортсме-нов. Кнопка «История тренировок» открывает модальное окно, а кнопка «Закрыть» и крестик в правом верхнем углу окна позволяет скрыть его. Такое поведение удобно реализовать с помощью хука useState.

```
import React , { useState } from ' react ' ; // ...
function SportsmanWin() {
const [showHistory, setShowHistory] = useState(false);
// ...
} // …
```

Первая строка в функции SportsmanWin() задает текущее состояние модального окна с историей тренировок спортсмена и функцию для изменения этого состояния. Изначально модальное окно скрыто, поэтому аргу- ментом в хук передается значение false. Окно должно появиться после нажатия на кнопку «История тренировок», в атрибутах кнопки указываем функцию изменения состояния:

```
onClick={() => setShowHistory(true)}
```

Атрибуты модального окна тоже будут зависеть от состояния этой переменной и функции изменения:

```
show={showHistory}
onHide={() => setShowHistory(false)}
```

Теперь модальное окно показывается в случае, когда showHistory имеет значение true. Таким образом, можно задавать не только состояния модальных окон, но и запускать различные эффекты после активации кнопкой, скрывать или показывать часть контента на странице при изменении текстового поля (поиск) или же сохранять значение текущего выбранного элемента (спортсмена или тренировки) на странице.

Еще один хук, о котором стоит упомянуть – useEffect. Этот хук запускает «эффект», определенную последовательность действий после изменения какого-либо состояния. В примере ниже с помощью хуков useState и useEffect реализован поиск.

```
import { sportsman } from ' . . / emulation ' // ...
function SportsmanWin() {
const [searchInput, setSearchInput] = useState( ' ');
const [searchResult, setSearchResult] = useState(sportsman); const handleChangeSearch = ( event ) => {
setSearchInput(event . target . value ); }
// ...
useEffect (() => {
const result = sportsman. filter (man =>
man.name.toLowerCase().includes(searchInput.toLowerCase()));
setSearchResult(result ); }, [ searchInput ])
return ( <FormControl
placeholder="Search"
value={searchInput}
onChange={handleChangeSearch} />)}
// ...
```

Задано изначально пустое состояние searchInput, в которое функция handleChangeSearch записывает введенную в текстовое поле строку при каждом его изменении. После каждого изменения этого состояния необходимо корректировать содержимое страницы, а именно

оставлять только те элементы, в которых содержится подстрока searchInput. Для этого прекрасно подойдет хук useEffect, принимающий первым аргументом стрелочную функцию, которая запускается каждый раз при изменении второго аргумента – массива состояний. В нашем случае, при каждом изменении текстового поля useEffect запускает функцию, которая формирует из массива объектов с информацией о спортсменах sportsman массив result, состоящий только из таких элементов массива sportsman, в поле name которых содержится подстрока searchInput.

В этом разделе были рассмотрены основные особенности разработки с помощью библиотеки React, а именно использование компонентов, настройка маршрутизации, использование хуков. Этих знаний было достаточно для разработки клиентской части приложения для оператора системы.

V. ЗАКЛЮЧЕНИЕ

В статье, выполненной в рамках выпускной квалификационной работы магистра, приведен взгляд авторов на процесс разработки интерфейса умного объекта на примере системы «Умный бассейн». Сформулированы принципы проектирования дизайна пользовательских интерфейсов, построены диаграмма сценариев использования приложения и диаграмма user-flow, выбран технологический стек с учетом современных тенденций frontend-разработки и особенностей разработки пользовательских интерфейсов с использованием библиотеки React.

Диаграмма сценариев использования приложения определяет тех, кто является пользователями системы, каковы их цели и какие шаги им нужно предпринять для достижения этой цели.

Диаграмма user-flow является визуальным представлением последовательности действий пользователей.

В качестве технологий для разработки пользовательского интерфейса выбран язык TypeScript и библиотека React, так как эти технологии обладают рядом преимуществ, позволяющих максимально быстро разрабатывать и повторно использовать элементы кода.

Подводя итог, можно сказать, что для разработки клиентских частей различных умных объектов лучше придерживаться шаблонного подхода к разработке, при котором система будет проще масштабироваться, если оперировать готовыми модулями и/или компонентами, а полученный интерфейс выглядит привычнее для пользователя. Также имеет смысл разрабатывать клиентскую часть так, чтобы вся нужная информация предоставлялась пользователю по мере необходимости, что не будет перегружать интерфейс, и сведет к минимуму нагрузку на кратковременную память пользователя. разрабатывать

БЛАГОДАРНОСТИ

Авторы выражают благодарность Высшей инжиниринговой школе НИЯУ МИФИ за помощь в возможности опубликовать результаты выполненной работы.

БИБЛИОГРАФИЯ

[1] Sanabria S., Jindal A. The Ifs and Buts of the Development Approaches for IoT Applications.—2021.—01.

[2] Dillon A. User Interface Design.—2006.—01.—ISBN: 9780470018866.

[3] Blair-Early A., Zender M. User Interface Design Principles for Interaction Design // Design Issues.—2008.—07.—Vol. 24.—P. 85–107.

[4] Glória A., Cercas F., Souto N. Design and implementation of an IoT gateway to create smart environments // Procedia Computer Science.—2017.—12.— Vol. 109.—P. 568–575.

[5] IoT-based intelligent fitness system / Yong B., Xu Z., Wang X., Cheng L., Li X.,Wu X., and Zhou Q. // Journal of Parallel and Distributed Computing.— 2017.—06.—Vol. 118.

[6] Landay J. A., Hong J. I. et al. The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience. — Addison-Wesley Professional, 2003.

[7] Fauzan R. et al. A Different Approach on Automated Use Case Diagram Semantic Assessment //International Journal of Intelligent Engineering and Systems. – 2021. – Т. 14. – №. 1. – С. 496-505.

[8] Knöps L. et al. Prototype Application for the Treatment of Achilles Tendinopathy. – 2021.

[9] Jermaniš E. Development of MESOC Toolkit Web Application in React : дис. – University of Rijeka. Department of Informatics, 2021.

[10] Boduch A., Derks R. React and React Native: A complete hands-on guide to modern web and mobile development with React. js. – Packt Publishing Ltd, 2020.

[11] Duldulao D. B., Cabagnot R. J. L. Navigating React Router //Practical Enterprise React. – Apress, Berkeley, CA, 2021. – С. 55-90.

[12] Bugl D. Learn React Hooks: Build and refactor modern React. js applications using Hooks. – Packt Publishing Ltd, 2019

Creating an interface for smart objects using the example of a smart swimming pool system

A.G. Klimenko, K.S. Zaytsev

***Annotation*. The development of smart objects involves solving a wide range of related tasks. One of these tasks is the development of the user interface. The purpose of this work is a consistent description of the process of developing the interface of a smart object using the example of the Smart Swimming Pool system. The basic principles of user interface development are determined, the scenario for using a web application is thought out, use-case and user-flow diagrams are built. Then development technologies are selected taking into account modern trends in frontend development and the workspace is set up. The results of choosing a technology stack, obtained by the authors, are presented in the form of a table of interface implementation tools. After that, the features of working with the React library are described, such as the ability to create your own or use built-in components, the ability to set up dynamic routing, the ability to use the current state of elements, and handling events using the useState and useEffect hooks. Code examples are provided to demonstrate all of these features.**

***Keywords* - Smart object, Internet of things, User interface, web application, use-case diagram, user-flow diagram, TypeScript, React**

REFERENCES

[1] Sanabria S., Jindal A. The Ifs and Buts of the Development Approaches for IoT Applications.—2021.—01.

[2] Dillon A. User Interface Design.—2006.—01.—ISBN: 9780470018866.

[3] Blair-Early A., Zender M. User Interface Design Principles for Interaction Design // Design Issues.—2008.—07.—Vol. 24.—P. 85–107.

[4] Glória A., Cercas F., Souto N. Design and implementation of an IoT gateway to create smart environments // Procedia Computer Science.—2017.—12.— Vol. 109.—P. 568–575.

[5] IoT-based intelligent fitness system / Yong B., Xu Z., Wang X., Cheng L., Li X.,Wu X., and Zhou Q. // Journal of Parallel and Distributed Computing.— 2017.—06.—Vol. 118.

[6] Landay J. A., Hong J. I. et al. The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience. — Addison-Wesley Professional, 2003.

[7] Fauzan R. et al. A Different Approach on Automated Use Case Diagram Semantic Assessment //International Journal of Intelligent Engineering and Systems. – 2021. – Т. 14. – №. 1. – С. 496-505.

[8] Knöps L. et al. Prototype Application for the Treatment of Achilles Tendinopathy. – 2021.

[9] Jermaniš E. Development of MESOC Toolkit Web Application in React : дис. – University of Rijeka. Department of Informatics, 2021.

[10] Boduch A., Derks R. React and React Native: A complete hands-on guide to modern web and mobile development with React. js. – Packt Publishing Ltd, 2020.

[11] Duldulao D. B., Cabagnot R. J. L. Navigating React Router //Practical Enterprise React. – Apress, Berkeley, CA, 2021. – С. 55-90.

[12] Bugl D. Learn React Hooks: Build and refactor modern React. js applications using Hooks. – Packt Publishing Ltd, 2019