

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, системы, комплексы и
сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

СИСТЕМА АВТОТЕСТИРОВАНИЯ САЙТОВ

Студент


(подпись, дата)

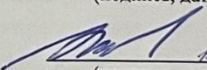
Осмаев Магомед-Али Хусейнович
(ФИО полностью)

Руководитель


(подпись, дата)

Митрохин М.А.
(фамилия, инициалы)

Нормоконтролёр


(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой


(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
19 июня 2026 г. № 14)

Секретарь ГЭК


(подпись)

Гурин Е.И.
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«07» 04 2022 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Осмаев Магомед-Али Хусейнович гр.22BBB1 факультета вычислительной техники.

2. Тема работы: Система автотестирования сайтов

Тема утверждена приказом ПГУ № 347/о от "7" апреля 2026 г.

3. Руководитель бакалаврской работы д.т.н кафедры ВТ ПГУ Митрохин М.А.

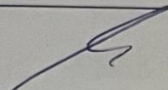
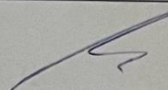
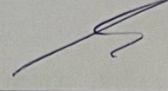
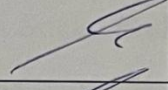
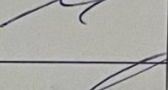
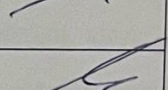
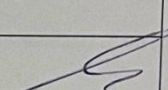
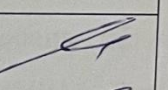
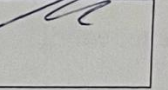
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать гибкую систему автотестирования веб-сайтов на базе Playwright и Visual Studio Code. Система должна обеспечивать возможность тестирования различных веб-приложений с минимальными изменениями конфигурации, поддерживать параметризацию тестов через механизм проектов Playwright, реализовывать паттерн Page Object Model, иметь модуль загрузки конфигурации из JS-файлов, а также включать собственное тестовое веб-приложение для демонстрации полного цикла.

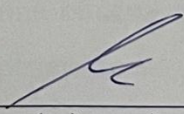
5. Перечень подлежащих разработке вопросов

Выполнить анализ инструментов автоматизации UI-тестирования и обосновать выбор Playwright; сформулировать требования к гибкой системе автотестирования; выбрать и настроить инструменты разработки; спроектировать архитектуру системы (конфигурационные файлы, модуль загрузки, Page Object); разработать тестовые сценарии для сайтов SauceDemo и OrangeHRM; создать собственное веб-приложение «Task Manager» и написать для него 5 автотестов; оценить гибкость системы по метрикам времени, объема кода и поддерживаемости; дать практические рекомендации.

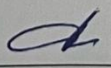
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объём работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области, анализ аналогов	30	07.04.2026 – 10.04.2026	
2	Выбор и настройка инструментов разработки	15	10.04.2026 – 14.04.2026	
3	Проектирование архитектуры системы	25	14.04.2026 – 18.04.2026	
4	Разработка конфигурационных файлов и модуля загрузки	30	18.04.2026 – 22.04.2026	
5	Реализация Page Object классов	35	22.04.2026 – 28.04.2026	
6	Создание собственного веб-приложения Task Manager	60	28.04.2026 – 08.05.2026	
7	Написание параметризованных тестов и отладка	40	08.05.2026 – 13.05.2026	
8	Тестирование системы на трёх сайтах	20	13.05.2026 – 16.05.2026	
9	Оценка эффективности, сравнение с монолитным подходом	20	16.05.2026 – 18.05.2026	
10	Оформление пояснительной записки	25	19.05.2026 – 23.05.2026	

Дата выдачи задания " 07 " апреля 2026г.

Руководитель бакалаврской работы  Митрохин М.А.
(подпись, дата)

Задание к исполнению принял студент  Осмаев М-А.Х.
(подпись, дата)

Работу к защите допустить
Декан факультета  19.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 103 листа, 22 рисунка, 5 таблиц, 20 источников, 3 приложения.

СИСТЕМА АВТОТЕСТИРОВАНИЯ САЙТОВ, PLAYWRIGHT, VISUAL STUDIO CODE, JAVASCRIPT, PAGE OBJECT MODEL, ГИБКАЯ АРХИТЕКТУРА, КОНФИГУРАЦИОННЫЕ ФАЙЛЫ, ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ, SAUCEDEMO, ORANGEHM, TASK MANAGER

Цель работы – разработать и экспериментально обосновать гибкую систему автотестирования веб-сайтов на базе Playwright и Visual Studio Code, позволяющую с минимальными изменениями тестировать различные веб-приложения, включая собственные проекты.

Объектом разработки является гибкая система автотестирования, которая включает: конфигурационные файлы для описания параметров сайтов, модуль загрузки конфигурации, реализацию паттерна Page Object, параметризованные тесты, а также собственное тестовое веб-приложение для демонстрации полного цикла.

В результате проделанной работы разработана и протестирована гибкая система автотестирования для трёх сайтов (SauceDemo, OrangeHRM, Task Manager). Написано 12 автотестов, все успешно выполняются. Проведена количественная оценка, подтверждающая сокращение времени подключения нового сайта в 4–6 раз и уменьшение объёма кода более чем вдвое. Даны практические рекомендации по внедрению системы.

					ПГУ 1.09.03 01.08.001 ПЗ			
Изм.	Лист	№ докум.	Под	Дата	Система автотестирования сайтов Пояснительная записка	Лит.	Лист	Листов
Разраб.		Осмаев М.-А.Х.		11.06.24			3	103
Провер.		Митрохин М.А.		11.06.24		ФВТ, каф. ВТ		
Н. Контр.		Бычков А.С.		11.06.24		гг. 22BBB1		
Утверд.		Митрохин М.А.		11.06.24				

Реферат

Бакалаврская работа содержит 103 листа, 22 рисунка, 5 таблиц, 20 источников, 3 приложения.

СИСТЕМА АВТОТЕСТИРОВАНИЯ САЙТОВ, PLAYWRIGHT, VISUAL STUDIO CODE, JAVASCRIPT, PAGE OBJECT MODEL, ГИБКАЯ АРХИТЕКТУРА, КОНФИГУРАЦИОННЫЕ ФАЙЛЫ, ПАРАМЕТРИЗОВАННЫЕ ТЕСТЫ, SAUCEDEMO, ORANGEHM, TASK MANAGER

Цель работы – разработать и экспериментально обосновать гибкую систему автотестирования веб-сайтов на базе Playwright и Visual Studio Code, позволяющую с минимальными изменениями тестировать различные веб-приложения, включая собственные проекты.

Объектом разработки является гибкая система автотестирования, которая включает: конфигурационные файлы для описания параметров сайтов, модуль загрузки конфигурации, реализацию паттерна Page Object, параметризованные тесты, а также собственное тестовое веб-приложение для демонстрации полного цикла.

В результате проделанной работы разработана и протестирована гибкая система автотестирования для трёх сайтов (SauceDemo, OrangeHRM, Task Manager). Написано 12 автотестов, все успешно выполняются. Проведена количественная оценка, подтверждающая сокращение времени подключения нового сайта в 4–6 раз и уменьшение объёма кода более чем вдвое. Даны практические рекомендации по внедрению системы.

					ПГУ 1.09.03.01.08.001 ПЗ			
Изм.	Лист	№ докум.	Под	Дата	Система автотестирования сайтов Пояснительная записка	Лит.	Лист	Листов
Разраб.		Осмаев М-А.Х.					3	103
Провер.		Митрохин М.А.				ФВТ, каф. ВТ гр. 22BBB1		
Н. Контр.		Бычков А.С.						
Утверд.		Митрохин М.А.						

Содержание

Перечень сокращений и обозначений	6
Введение	8
1 Система автотестирования сайтов	11
1.1 Анализ предметной области	11
1.1.1 Анализ аналогов	13
1.1.2 Постановка задачи	16
1.2 Выбор и настройка инструментов разработки	19
1.2.1 Критерии выбора инструментов	19
1.2.2 Основной инструмент: Playwright	20
1.2.3 Язык программирования: JavaScript	21
1.2.4 Среда разработки: Visual Studio Code	22
1.2.5 Среда выполнения и управление пакетами: Node.js + npm	22
1.2.6 Пошаговая настройка рабочего окружения	23
2 Практическая реализация	30
2.1 UML-диаграммы архитектуры системы	30
2.2 Разработка универсальных тестовых сценариев	37
2.3 Создание собственного веб-приложения «Task Manager»	39
2.4 Реализация гибкой архитектуры	42
2.5 Описание тестовых веб-приложений	53
2.6 Запуск тестов и полученные результаты	56
2.7 Оценка гибкости и сравнение с «монолитным» подходом	58
3 Безопасность жизнедеятельности	60
3.1 Требования к организации рабочего места	60
3.2 Требование к освещению	62
3.3 Требования к режиму отдыха	63
Заключение	65
Список использованных источников	67

Приложение А – UML-диаграммы	69
Приложение Б – Листинг программы	74
Приложение В – Презентация	98

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Перечень сокращений и обозначений

API (Application Programming Interface) – программный интерфейс приложения

CI/CD (Continuous Integration / Continuous Delivery) – непрерывная интеграция и непрерывная доставка

Codegen – инструмент Playwright для автоматической генерации кода тестов записью действий

DOM – структура страницы в виде дерева элементов

ESLint - инструмент статического анализа кода JavaScript для выявления ошибок

HTML (HyperText Markup Language) – язык гипертекстовой разметки

HTTP (HyperText Transfer Protocol) – протокол передачи гипертекста
UI – User Interface (пользовательский интерфейс)

IDE (Integrated Development Environment) – интегрированная среда разработки

JS (JavaScript) - язык программирования, используемый для веб-разработки и написания тестов

JSON (JavaScript Object Notation) – текстовый формат обмена данными

LTS (Long Term Support) – версия программного обеспечения с длительной поддержкой

Npm (Node Package Manager) – менеджер пакетов для Node.js

Npx (Node Package Executor) – исполнитель пакетов Node.js

OrangeHRM - тестовое веб-приложение – система управления персоналом

Page Object Model (POM) - паттерн проектирования, при котором для каждой страницы создаётся отдельный класс с элементами и методами

Playwright - фреймворк для автоматизации тестирования веб-приложений от Microsoft

Prettier - инструмент автоматического форматирования кода

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

SauceDemo - учебный интернет-магазин, созданный для обучения автоматизации тестирования

SPA (Single Page Application) – одностраничное приложение

TaskManager - собственное веб-приложение, разработанное в рамках дипломной работы

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

VS Code (Visual Studio Code) – редактор исходного кода от Microsoft

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Современная веб-разработка характеризуется короткими циклами релизов, частыми изменениями пользовательского интерфейса и высокими требованиями к качеству программных продуктов. Методологии гибкой разработки (Agile, Scrum, Kanban) предполагают выпуск новых версий каждые несколько дней или даже часов. В этих условиях ручное регрессионное тестирование становится узким местом: оно трудоёмко, подвержено человеческим ошибкам и не успевает за скоростью разработки. По данным исследований, автоматизация тестирования позволяет сократить время регрессионного прогона на 70–80% при условии правильного выбора инструментов и архитектуры тестового фреймворка.

Особую актуальность приобретают системы автотестирования, которые не привязаны к одному конкретному сайту, а обладают гибкостью, позволяющей тестировать различные веб-приложения с минимальными доработками. На рынке инструментов автоматизации UI-тестирования в последние годы произошли значительные изменения. Долгое время стандартом де-факто был Selenium WebDriver, однако он обладает рядом недостатков: сложная настройка ожиданий, отсутствие встроенной записи сценариев, необходимость использования дополнительных библиотек. В ответ на эти ограничения появились современные инструменты — Cypress и Playwright, которые предлагают лучший опыт разработки и встроенные механизмы генерации тестов.

Особый интерес представляет инструмент Playwright от Microsoft, который с момента выхода в 2020 году быстро набирает популярность. Согласно опросу State of JS 2024, удовлетворённость Playwright составляет 92%, что выше, чем у Cypress (86%) и Selenium (78%). Ключевые преимущества Playwright — встроенные автоожидания, поддержка всех современных браузеров (Chromium, Firefox, WebKit) и мощный инструмент

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		8

записи сценариев Codegen. Однако остаётся открытым вопрос: как построить гибкую систему автотестирования, которая позволяла бы легко переключаться между разными сайтами, поддерживала конфигурируемые локаторы и сценарии, и при этом обеспечивала высокую поддерживаемость тестов. Данная работа направлена на решение этой проблемы.

Цель дипломной работы — разработка и экспериментальное обоснование гибкой системы автотестирования сайтов на базе Playwright и Visual Studio Code, позволяющей с минимальными изменениями тестировать различные веб-приложения, включая собственные проекты.

В ходе выполнения дипломной работы необходимо провести анализ предметной области автоматизации UI-тестирования и существующих инструментов (Selenium WebDriver, Cypress, Playwright), обосновав выбор Playwright. Следует сформулировать требования к гибкой системе автотестирования, не привязанной к одному сайту, а затем выбрать и настроить инструменты разработки: Playwright, Visual Studio Code, Node.js, JavaScript. Далее требуется разработать архитектуру системы, включающую конфигурационные файлы для разных сайтов, модуль работы с локаторами и реализацию паттерна Page Object Model. На основе этой архитектуры необходимо создать набор типовых тестовых сценариев, демонстрирующих гибкость системы на примере двух различных веб-приложений (SauceDemo и OrangeHRM). Параллельно следует разработать собственное тестовое веб-приложение (менеджер задач) и написать для него автотесты. Завершающим этапом станет оценка разработанной системы по таким метрикам, как время настройки для нового сайта, объём дублирования кода и поддерживаемость, а также формулировка практических рекомендаций по использованию системы в реальных проектах. Методы исследования: анализ научно-технической литературы и документации, сравнительный анализ инструментов,

эксперименты с программированием, количественная оценка метрик (время разработки, объём кода, число правок при изменении вёрстки).

Практическая значимость работы. Разработанная система автотестирования может быть использована:

- в IT-компаниях и стартапах для быстрого развёртывания автоматизированного тестирования разных веб-сайтов;

- в учебном процессе при изучении курсов «Тестирование программного обеспечения» и «Автоматизация тестирования»;

- как база для дальнейших исследований (интеграция с CI/CD, визуальное регрессионное тестирование, кросс-браузерное тестирование).

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

1 Система автотестирования сайтов

1.1 Анализ предметной области

Предметная область данной работы — автоматизация тестирования пользовательского интерфейса (UI) веб-приложений с ориентацией на создание гибкой системы, которая может быть адаптирована для тестирования различных сайтов без переписывания основной логики тестов.

Под автоматизацией UI-тестирования понимается процесс создания программных сценариев, которые имитируют действия реального пользователя (клики, ввод текста, навигация) и проверяют корректность работы веб-приложения. Гибкость системы означает:

- возможность легко переключаться между разными тестовыми сайтами (изменение базового URL, уникальных селекторов, тестовых данных);
- минимальные изменения в коде тестов при добавлении нового сайта;
- централизованное управление локаторами, тестовыми данными и конфигурацией;
- поддержка различных типов веб-приложений (интернет-магазины, корпоративные порталы, одностраничные приложения).

Основные требования к гибкой системе автотестирования представлены в таблице 1.

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1 – Требования к гибкой системе автотестирования

Требование	Описание
Независимость от сайта	Система не должна содержать жёстко зашитых URL, селекторов и данных конкретного сайта. Вся сайто-специфичная информация выносится в конфигурацию.
Конфигурируемость	Возможность задавать для каждого сайта: набор страниц, локаторы элементов, учётные данные, сценарии проверки.
Поддерживаемость	При изменении вёрстки одного сайта правки вносятся только в конфигурацию этого сайта, не затрагивая другие.
Модульность	Повторяющиеся действия (логин, поиск элемента, ожидание) реализованы в виде переиспользуемых функций/классов.
Расширяемость	Лёгкое добавление новых сайтов и новых типов тестов (например, визуальное регрессионное тестирование).
Документированность	Система должна иметь понятную структуру и инструкцию по подключению нового сайта.

На практике многие компании создают автотесты, жёстко привязанные к одному приложению. При переходе на другой проект тесты либо переписываются, либо адаптируются с большими трудозатратами. Поэтому разработка универсальной, легко настраиваемой системы автотестирования является актуальной научно-практической задачей.

1.1.1 Анализ аналогов

Для обоснованного выбора инструмента реализации системы автотестирования рассмотрим основные аналоги, представленные на рынке автоматизации UI-тестирования. Ключевые критерии сравнения: поддержка браузеров, наличие автоожиданий, запись сценариев, параллельный запуск, гибкость настройки под разные сайты, стоимость.

Selenium WebDriver — это стандарт в области автоматизации UI-тестирования, существующий с 2008 года. Инструмент поддерживает все основные браузеры (Chrome, Firefox, Safari, Edge) и языки программирования (Java, Python, C#, Ruby, JavaScript).

Недостатки: отсутствие встроенных автоожиданий (разработчик вынужден вручную добавлять явные или неявные ожидания), нет встроенной записи сценариев (Selenium IDE устарел и не интегрируется с WebDriver). Для тестирования разных сайтов требуется писать много шаблонного кода, переключать драйверы.

Достоинства: огромное сообщество, обилие документации. Однако для построения гибкой системы Selenium не предоставляет удобных механизмов «из коробки».

Cypress — современный инструмент, появившийся в 2017 году, ориентированный на frontend-разработчиков.

Преимущества: встроенные автоожидания, удобный Time Travel (возможность видеть каждый шаг выполнения), отличная документация.

Ограничением является поддержка только Chromium-браузеров (Chrome, Edge), невозможность работы с несколькими вкладками, бесплатная запись сценариев отсутствует (Cypress Studio — платная). Параллельный запуск также требует платной подписки. С точки зрения гибкости для разных сайтов — Cypress позволяет выносить конфигурацию, но его привязка к одному домену (ограничения политики same-origin) затрудняет тестирование сайтов с переходами на внешние ресурсы.

Playwright разработан компанией Microsoft, первая стабильная версия выпущена в 2020 году.

Ключевые преимущества: поддержка всех браузеров (Chromium, Firefox, WebKit), встроенные автоожидания, параллельный запуск из коробки, работа с несколькими вкладками и эмуляция мобильных устройств. Имеется бесплатный инструмент записи сценариев Codegen. Поддерживает JavaScript, TypeScript, Python, C# и Java. Высокая скорость выполнения.

Гибкость Playwright позволяет легко параметризовать тесты через конфигурационные файлы, переменные окружения, а также предоставляет API для динамического изменения локаторов и базовых URL. Благодаря встроенному механизму fixtures можно создавать переиспользуемые наборы действий для разных сайтов.

Сравнительная таблица аналогов (таблица 2).

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 2 – Сравнение инструментов автоматизации UI-тестирования

Критерий	Selenium WebDriver	Cypress	Playwright
Поддержка браузеров	Все	Только Chromium	Все (Chromium, Firefox, WebKit)
Автоожидания	Нет	Да	Да
Запись сценариев (бесплатно)	Нет (IDE устарел)	Нет (только платно)	Да (Codegen)
Параллельный запуск	Требует Grid	Только платно	Из коробки
Несколько вкладок	Сложно	Не поддерживается	Да
Мобильные устройства	Через Appium	Частично	Да
Языки программирования	5+	Только JS/TS	5+
Удобство настройки под разные сайты	Среднее (всё вручную)	Низкое (ограничения домена)	Высокое (конфигурация, fixtures)

Критерий	Selenium WebDriver	Cypress	Playwright
Скорость выполнения	Средняя	Высокая	Высокая
Стоимость	Бесплатно	Freemium	Бесплатно
Удовлетворённость разработчиков (2024)	78%	86%	92%

Вывод по анализу аналогов. Selenium WebDriver остаётся жизнеспособным для проектов с большим наследием, но не рекомендуется для новых систем из-за отсутствия автоожиданий и низкой гибкости. Cypress подходит для одностраничных приложений (SPA), но его ограничения по браузерам и доменной модели делают его менее универсальным. Playwright является оптимальным выбором для построения гибкой системы автотестирования сайтов, так как он сочетает высокую производительность, встроенные механизмы записи, мощный API для конфигурации и полную бесплатность.

1.1.2 Постановка задачи

На основе проведённого анализа предметной области и сравнения аналогов были сформулированы функциональные и технические требования к разрабатываемой гибкой системе автотестирования.

Функциональные требования определяют, какие возможности система предоставляет конечному пользователю. Для наглядного представления этих требований построена диаграмма вариантов использования (рисунок 1). Как показано на диаграмме, в системе выделяются два актёра: «Тестировщик» и

«Разработчик сайта». Тестировщик имеет возможность настраивать конфигурацию сайта, то есть создавать или редактировать JS-модули с базовым URL, учётными данными и селекторами элементов. Он может запускать тесты как для конкретного сайта (выбирая соответствующий проект), так и для всех проектов, определённых в конфигурации Playwright. После выполнения тестов система предоставляет HTML-отчёт, а в случае падения позволяет просмотреть трассировку и видео прохождения теста. Кроме того, тестировщик может писать новые Page Object'ы и разрабатывать универсальные параметризованные тесты, которые автоматически подхватывают конфигурацию сайта. Разработчик сайта, в свою очередь, может добавлять в свой проект атрибуты data-test, что повышает устойчивость тестов и упрощает их поддержку.

На рисунке 1 представлена диаграмма вариантов использования, которая моделирует функциональные требования к разрабатываемой гибкой системе автотестирования.

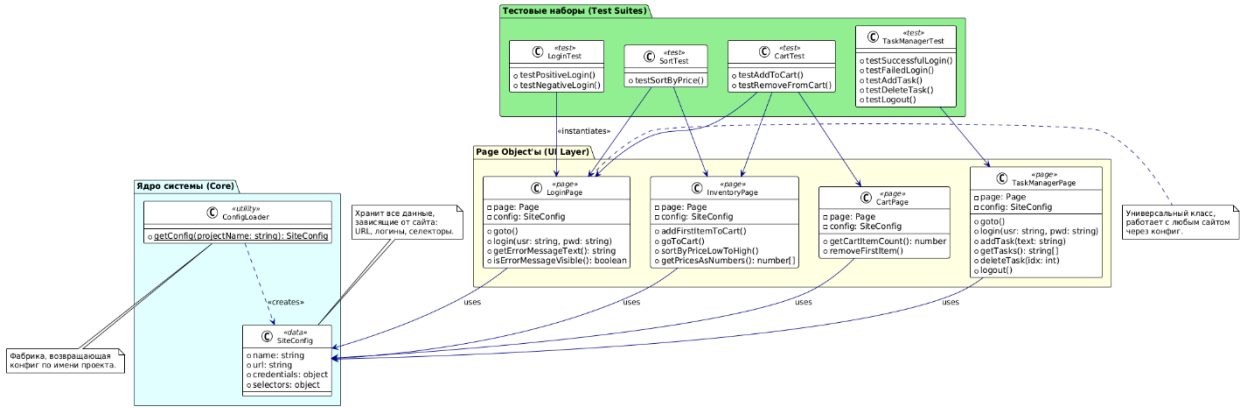


Рисунок 1 – Диаграмма вариантов использования

Технические требования определяют, на каких инструментах и с использованием каких архитектурных решений должна быть реализована система. В качестве языка реализации выбран JavaScript (стандарт ES2022), а в качестве движка автоматизации – фреймворк Playwright. Среда разработки – Visual Studio Code. Архитектура системы строится на принципе разделения

данных и логики: для каждого тестируемого сайта создаётся отдельный конфигурационный файл, содержащий имя сайта, базовый URL, учётные данные и CSS-селекторы элементов. Логика тестов и Page Object'ов не обращается к этим параметрам напрямую, а получает их через специальный модуль загрузки. Тесты параметризуются с помощью механизма проектов Playwright, где каждый проект соответствует одному сайту и имеет собственный baseUrl и маску testMatch. Обязательным является реализация паттерна Page Object: универсальные классы (например, LoginPage) работают с любым сайтом, получая селекторы из конфигурации, а специфичные классы (например, InventoryPage или TaskManagerPage) создаются для уникального функционала отдельных сайтов. Учётные данные не должны храниться в коде – для них используются переменные окружения (файл .env с загрузкой через dotenv). Система должна автоматически формировать HTML-отчёты, а в случае падения теста – сохранять скриншоты, видео и трассировку. Архитектура должна быть масштабируемой: добавление нового сайта не должно требовать изменения существующих тестов – достаточно создать новый конфигурационный файл и, при необходимости, добавить специфичные Page Object'ы. Для демонстрации полного цикла разработки и отладки система должна включать собственное тестовое веб-приложение «Task Manager», реализованное на чистом HTML, CSS и JavaScript с хранением данных в localStorage.

Ожидаемыми результатами работы являются исходный код системы автотестирования на JavaScript/Playwright, конфигурационные файлы для трёх тестовых сайтов (SauceDemo, OrangeHRM, TaskManager), набор параметризованных тестов (не менее 12 сценариев) с успешным их выполнением на каждом сайте, сравнительная таблица метрик «жёсткий vs гибкий подход», а также текстовые рекомендации по внедрению системы в реальных проектах.

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Выбор и настройка инструментов разработки

Для построения гибкой системы автотестирования необходимо обоснованно выбрать и настроить комплекс инструментов, обеспечивающих написание, запуск и отладку тестов, а также управление конфигурациями. В данном разделе подробно рассматриваются критерии выбора, обоснование каждого инструмента и пошаговая инструкция по настройке рабочего окружения с примерами кода.

1.2.1 Критерии выбора инструментов

При выборе инструментов разработки учитывались как обязательные, так и желательные критерии. К числу обязательных отнесены бесплатность (возможность использования в коммерческих и учебных целях без лицензионных отчислений), кроссплатформенность (поддержка Windows, macOS и Linux), наличие активной поддержки и сообщества, совместимость инструментов друг с другом, а также поддержка гибкой конфигурации — возможность переключать сайты, локаторы и тестовые данные без изменения кода тестов. В качестве дополнительных, но желательных критериев рассматривались встроенные автоожидания, сокращающие время разработки и повышающие стабильность тестов, наличие механизма записи сценариев для быстрого прототипирования при подключении нового сайта, параллельный запуск, ускоряющий выполнение тестов на нескольких сайтах, а также развитые средства отладки, включающие трассировку, видео, скриншоты и пошаговое выполнение.

1.2.2 Основной инструмент: Playwright

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						19
Изм.	Лист	№ докум.	Подпись	Дата		

Playwright (Microsoft, стабильный релиз с 2020 года) выбран в качестве единственного движка автоматизации по следующим ключевым причинам, важным именно для гибкой системы.

1.2.2.1 Поддержка всех браузеров и унификация API

Playwright из коробки работает с Chromium (Chrome, Edge), Firefox и WebKit (Safari). Это критично для гибкой системы: разные сайты могут предъявлять разные требования к браузерам (например, корпоративный портал может быть оптимизирован только под Chrome, а интернет-магазин — под Safari). В Selenium для смены браузера нужно менять драйвер и настройки; в Cypress смена браузера ограничена Chromium-семейством. Playwright позволяет одной командой запустить один и тот же тест во всех браузерах.

В отличие от Selenium, где необходимо вручную писать WebDriverWait, Playwright автоматически ожидает, пока элемент станет видимым, стабильным (не движется) и получит возможность принять действие (например, клик). Это реализовано через механизм actionability checks. Для гибкой системы это означает, что при переключении на другой сайт с другой скоростью загрузки не нужно переписывать ожидания – Playwright адаптируется динамически.

Codegen позволяет генерировать код теста записью действий в браузере. Для гибкой системы это даёт возможность:

- быстро получить рабочие локаторы для нового сайта (даже если у сайта нет data-test атрибутов);
- создать прототип теста за 2–3 минуты, а затем перенести сгенерированные селекторы в конфигурационный файл;
- обучать новых членов команды.

Playwright использует единый конфигурационный файл playwright.config.js, в котором можно определить несколько проектов (projects) – каждый проект описывает браузер, базовый URL, окружение и

другие параметры. Для гибкой системы это идеальный механизм: один проект = один тестируемый сайт. Кроме того, Playwright поддерживает фикстуры (fixtures) – механизм внедрения зависимостей, позволяющий создавать переиспользуемые объекты (например, загруженную конфигурацию конкретного сайта) и передавать их в тесты.

В реальных сценариях пользователь часто открывает ссылки в новых вкладках, переключается между ними. Cypress не поддерживает работу с несколькими вкладками, Selenium требует сложного переключения windowHandles. Playwright предоставляет простой API через context.pages() и page.waitForEvent('popup').

Playwright позволяет подменять ответы сервера, эмулировать офлайн-режим, блокировать сторонние скрипты. Это полезно для гибкой системы: можно тестировать сайт в изоляции, не полагаясь на внешние API.

При падении теста Playwright автоматически сохраняет скриншот, видео прохождения и трассировку (trace) – интерактивный отчёт с хронологией действий, вызовами API, консольными сообщениями. Это радикально упрощает анализ ошибок, особенно когда тесты запускаются на удалённом CI.

Playwright распространяется под лицензией Apache 2.0, полностью бесплатен.

1.2.3 Язык программирования: JavaScript

Для реализации системы выбран JavaScript современного стандарта (ES2022). Обоснование:

- нативная поддержка Playwright. Вся официальная документация, примеры и API ориентированы в первую очередь на JavaScript/TypeScript;
- отсутствие этапа компиляции. Код пишется и сразу выполняется, что ускоряет итерации при настройке нового сайта;
- простота синтаксиса. Даже тестировщик с базовыми знаниями программирования может понимать и модифицировать тесты;

– конфигурационные файлы как модули. Благодаря тому, что конфигурации сайтов оформляются в виде JS-файлов, в них можно использовать вычисляемые значения, условия, импортировать общие части.

TypeScript добавляет статическую типизацию, что полезно для крупных проектов. Однако в данной дипломной работе приоритетом является наглядность и минимальный порог входа. Переход на TypeScript может быть выполнен позже без изменения логики.

В коде используются современные возможности: `async/await`, стрелочные функции, деструктуризация, модули `import/export`.

1.2.4 Среда разработки: Visual Studio Code

Visual Studio Code — бесплатный, легковесный и мощный редактор от Microsoft. Причины выбора:

- интеграция с Playwright. Официальное расширение «Playwright Test for VSCode» добавляет подсветку синтаксиса, автодополнение, позволяет запускать и отлаживать тесты прямо из редактора;

- встроенный терминал и Git. Не нужно переключаться между окнами для выполнения команд;

- расширяемость. Легко устанавливаются плагины для форматирования кода (Prettier), статического анализа (ESLint), работы с JSON и т.д.;

- кроссплатформенность. Работает на всех основных ОС.

Альтернативы (WebStorm, Sublime Text) либо платные, либо требуют больше ручной настройки.

1.2.5 Среда выполнения и управление пакетами: Node.js + npm

Node.js (версия 20.x LTS) является средой выполнения JavaScript вне браузера. Она необходима для работы Playwright, который поставляется как пакет npm. Выбор версии 20.x обусловлен длительной поддержкой (LTS) и полной совместимостью с Playwright.

Npm (Node Package Manager) используется для:

- установки Playwright: `npm init playwright@latest`;
- добавления вспомогательных библиотек (например, `@faker-js/faker` для генерации тестовых данных, `dotenv` для работы с переменными окружения);
- управления скриптами запуска, описанными в `package.json`.

1.2.6 Пошаговая настройка рабочего окружения

Ниже приведена подробная инструкция, учитывающая создание гибкой системы с поддержкой нескольких сайтов. Все действия выполняются в командной строке (терминале). Инструкция разбита на логические шаги с пояснениями.

Шаг 1. Установка Node.js (версия 20.x LTS):

- перейдите на официальный сайт Node.js: <https://nodejs.org/>;
- скачайте установщик для вашей операционной системы с пометкой «LTS» (рекомендованная версия). На момент написания работы — 20.20.2.;
- запустите установщик. Настройки оставьте по умолчанию. Обязательно установите `npm` (включён по умолчанию);
- после завершения установки перезагрузите компьютер (необходимо для корректного добавления переменных окружения);
- проверьте успешность установки, открыв командную строку (терминал) и выполнив:

а) `node --version` # должно вывести `v20.x.x`;

б) `npm --version` # должно вывести номер версии `npm` (например, `10.x.x`).

На рисунке 2 показан результат проверки успешной установки Node.js и менеджера пакетов `npm`. Команды `node --version` и `npm --version` выводят установленные версии, что подтверждает корректность настройки среды выполнения перед началом разработки. Для данной работы использовалась версия Node.js 20.x LTS.

```
D:\DIPLOM>node --version
v20.20.2

D:\DIPLOM>npm --version
10.8.2
```

Рисунок 2 – Проверка версий

На рисунках 3–5 отображены этапы установки интегрированной среды разработки Visual Studio Code. Процесс включает загрузку установочного файла с официального сайта, запуск инсталлятора и выбор параметров установки.

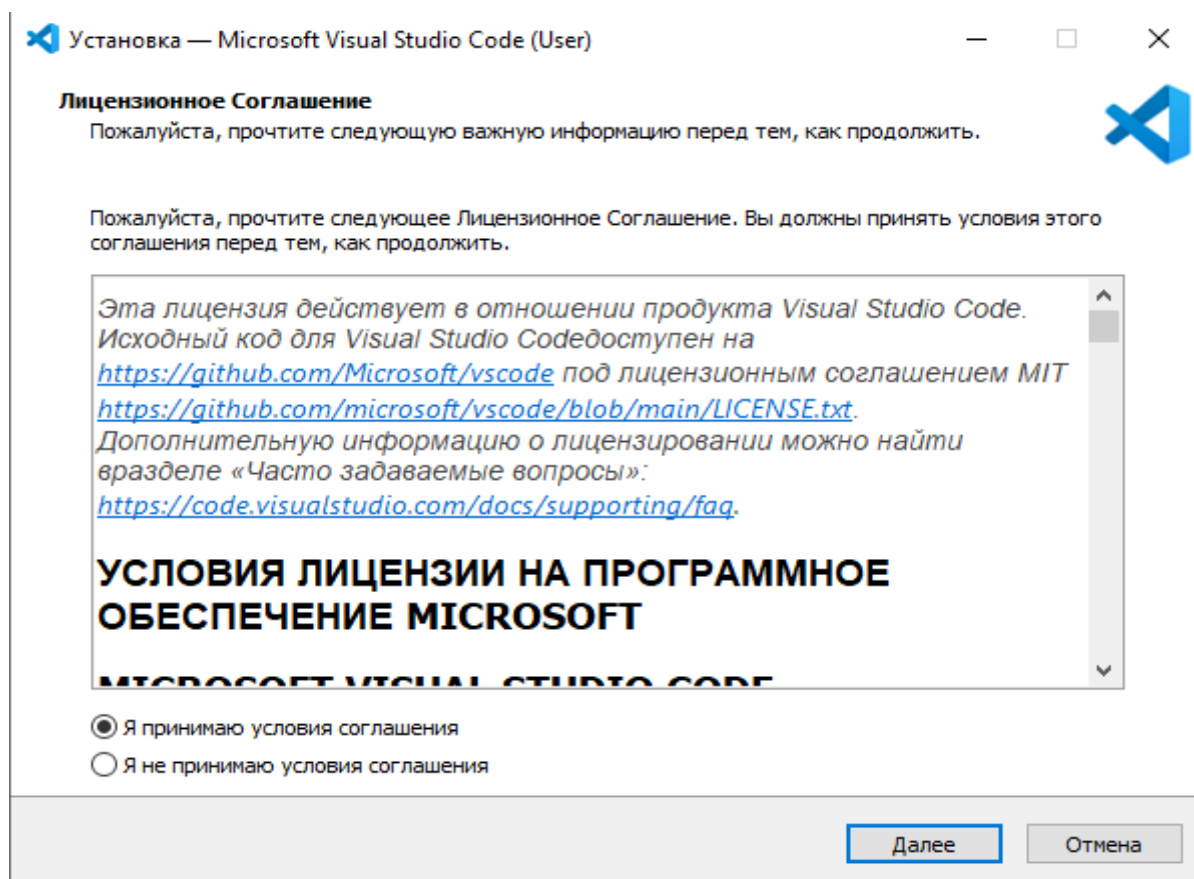


Рисунок 3 – Установка IDE

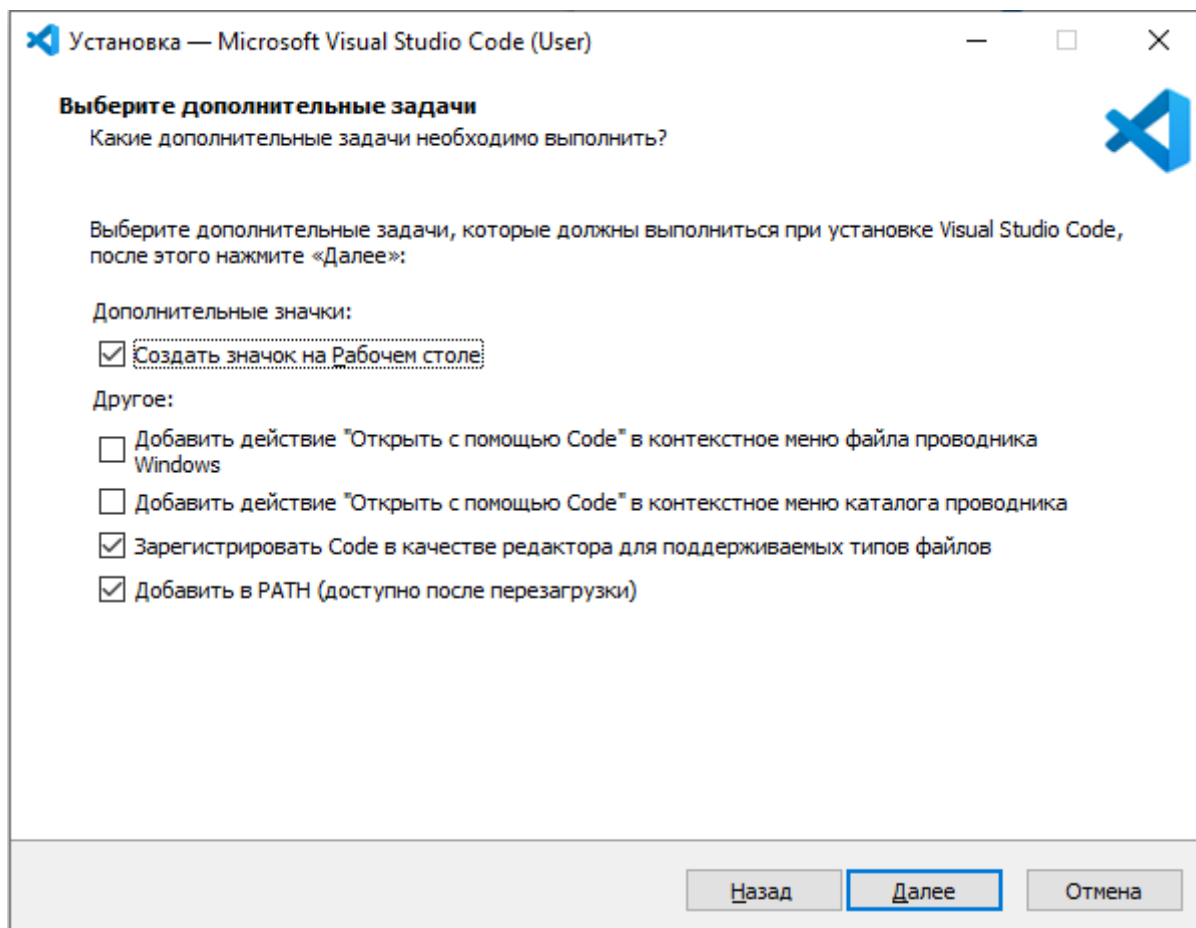


Рисунок 4 – Установка IDE



Рисунок 5 – Установка IDE

Шаг 2. Создание папки для проекта:

– в командной строке выполните:

а) `mkdir flexible-autotest`;

б) `cd flexible-autotest`.

Эти команды создадут новую папку `flexible-autotest` и перейдут в неё.

Шаг 3. Установка Playwright и браузеров:

– выполните команду:

`Npm init playwright@latest`.

– интерактивно ответьте на вопросы (рекомендуемые ответы):

а) `choose language` → JavaScript (или TypeScript, но для данной работы выбран JS);

б) `name of your test directory` → tests (оставить по умолчанию);

в) `add GitHub Actions?` → No (для дипломной работы не требуется);

г) `install Playwright browsers?` → Yes (будет загружено ~300 МБ браузеров).

– после завершения установки в папке появятся:

а) `node_modules/` — все зависимости;

б) `tests/` — пример теста `example.spec.js`;

в) `playwright.config.js` — главный конфигурационный файл;

г) `package.json` — дополнен зависимостями и скриптами.

Рисунок 6 демонстрирует процесс установки Playwright с помощью команды `npm init playwright@latest`. На скриншоте показаны интерактивные ответы на вопросы: выбор языка JavaScript, указание директории для тестов (tests), отказ от добавления GitHub Actions и подтверждение установки браузеров (Chromium, Firefox, WebKit).

```
Getting started with writing end-to-end tests with Playwright:
Initializing project in '.'
✓ Do you want to use TypeScript or JavaScript? · JavaScript
✓ Where to put your end-to-end tests? · tests
✓ Add a GitHub Actions workflow? (Y/n) · false
✓ Install Playwright browsers (can be done manually via 'npx playwright install')? (Y/n) · true
Installing Playwright Test (npm install --save-dev @playwright/test)...
```

Рисунок 6 - Установка Playwright и браузеров

Шаг 4. Создание структуры каталогов для гибкой системы:

Организуйте папки согласно архитектуре.

На рисунке 7 представлена структура каталогов проекта после выполнения команды `mkdir config config\sites pages utils`. Созданы основные папки: `config/sites` для конфигурационных файлов сайтов, `pages` для Page Object классов, `tests` для тестов и `utils` для вспомогательных модулей.

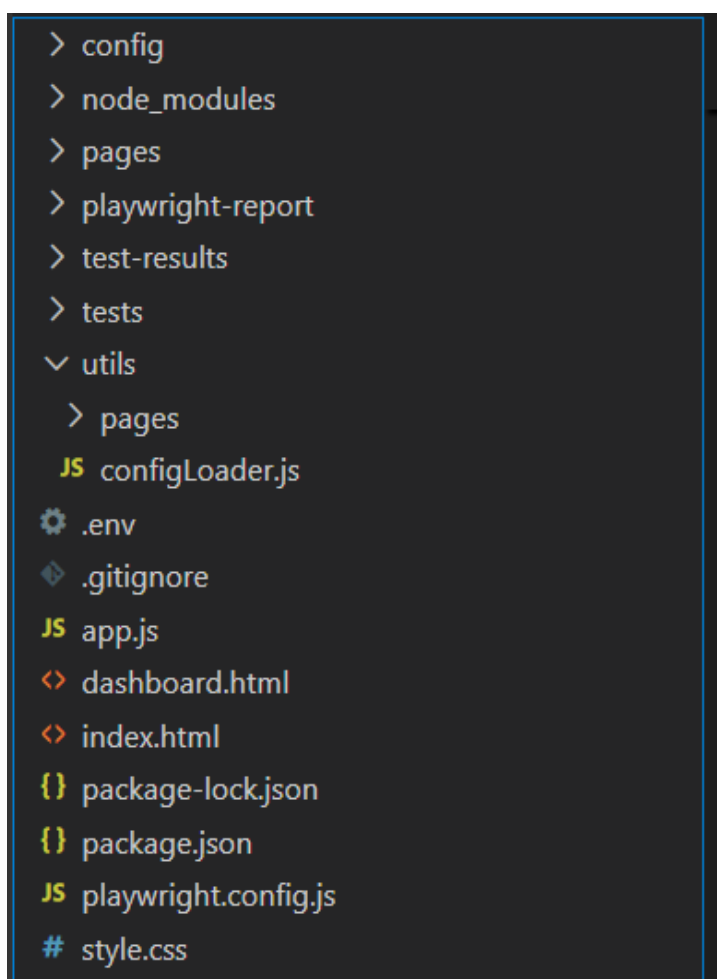


Рисунок 7 - структура каталогов

Шаг 5. Создание конфигурационных файлов для сайтов:

- в папке config/sites/ создайте файл saucedemo.config.js (содержимое приведено в разделе 2.3.1);
- создайте файл orangehrm.config.js и taskmanager.config.js;
- каждый файл экспортирует объект с полями name, url, selectors.

Рисунок 8 показывает созданные конфигурационные файлы для трёх тестируемых

сайтов: saucedemo.config.js, orangehrm.config.js и taskmanager.config.js. Каждый файл экспортирует объект с полями name, url, credentials и selectors

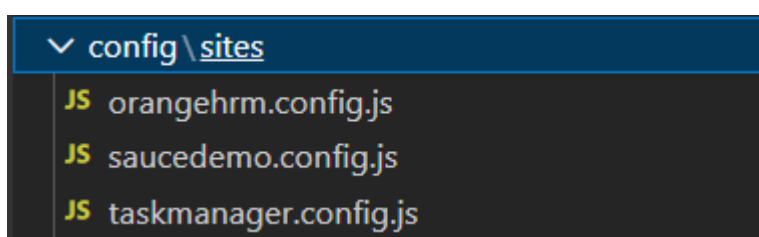


Рисунок 8 - конфигурационные файлы

Шаг 6. Установка расширений Visual Studio Code:

- откройте VS Code в папке проекта (code .);
- перейдите в раздел расширений (иконка квадратиков или Ctrl+Shift+X);
- установите следующие расширения:
 - а) Playwright Test for VSCode (Microsoft) – для запуска и отладки тестов;
 - б) Prettier – Code formatter – автоматическое форматирование кода;
 - в) ESLint – поиск синтаксических ошибок и стилистических проблем.
- после установки перезагрузите VS Code.

На рисунках 9–11 показан процесс установки расширений для Visual Studio Code: Playwright Test for VSCode (для запуска и отладки тестов), Prettier – Code formatter (для автоматического форматирования кода) и ESLint (для статического анализа JavaScript).

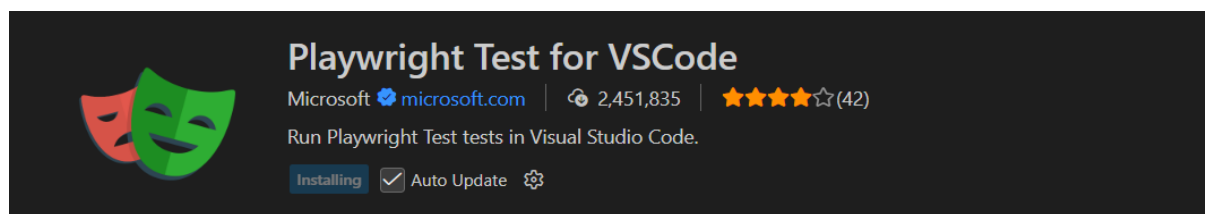


Рисунок 9 – Установка расширения для VS Code

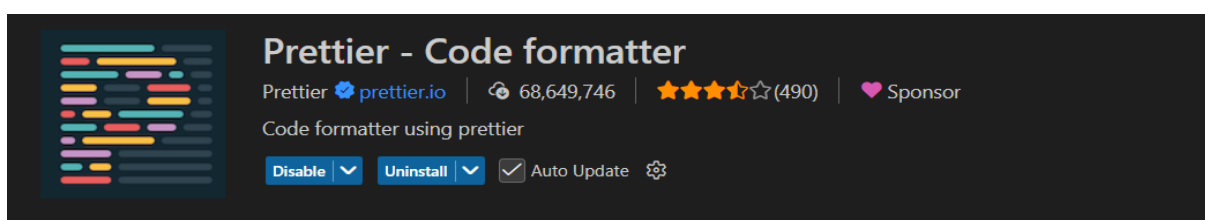


Рисунок 10 – Установка расширения для VS Code

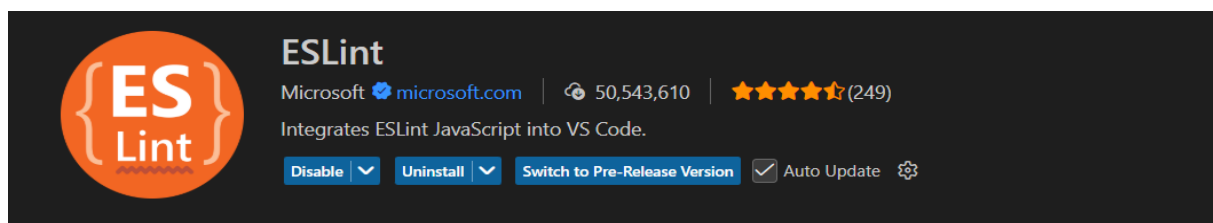


Рисунок 11 – Установка расширения для VS Code

2 Практическая реализация

2.1 UML-диаграммы архитектуры системы

Для формализации и наглядного представления разработанной гибкой системы автотестирования были построены четыре основные диаграммы унифицированного языка моделирования (UML): диаграмма вариантов использования, диаграмма классов, диаграмма последовательности и диаграмма компонентов. Каждая из диаграмм раскрывает определённый аспект системы: функциональные требования, статическую структуру, динамику взаимодействия и физическое развёртывание. Все диаграммы выполнены в едином стиле, снабжены легендами и цветовым кодированием для облегчения восприятия.

Диаграмма вариантов использования (рисунок 12) моделирует функциональные требования к системе с точки зрения внешних пользователей (актёров). На диаграмме выделены два актёра: «Тестировщик» и «Разработчик сайта» (изображены в виде стилизованных человечков, что соответствует стандарту UML для актёров).

Тестировщик выполняет следующие прецеденты (варианты использования):

- «настроить конфигурацию сайта» – создание или редактирование конфигурационного файла для нового тестируемого сайта (URL, учётные данные, селекторы);

- «запустить тесты для конкретного сайта» – выполнение тестов только для одного из подключённых проектов;

- «запустить все тесты» – выполнение всех тестов для всех проектов, определённых в playwright.config.js.;

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

– «получить HTML-отчёт» – формирование детального отчёта о результатах тестирования;

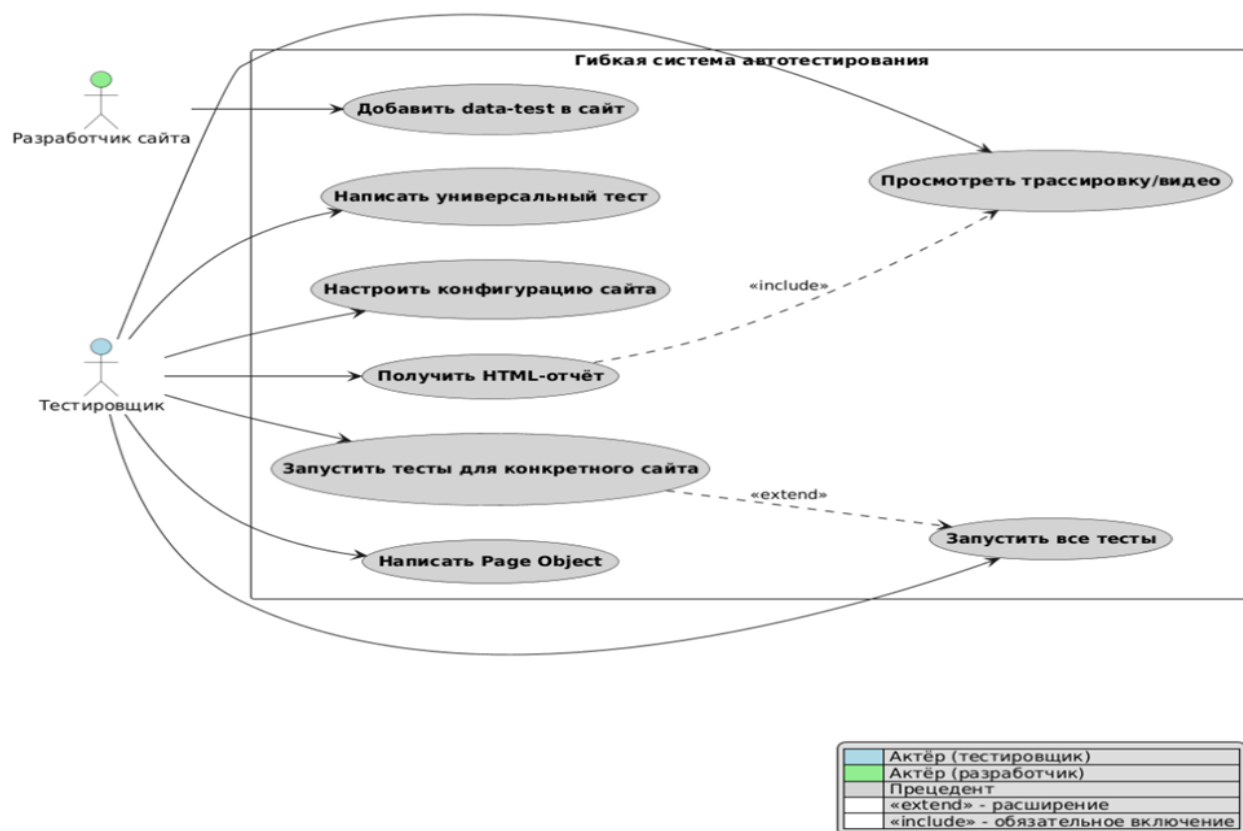


Рисунок 12 – Диаграмма вариантов использования (Use Case Diagram)

– «просмотреть трассировку/видео» – анализ записанных действий браузера и сетевых запросов при падении тестов (отношение <<include>> к получению отчёта);

– «написать Page Object» – создание нового класса, инкапсулирующего взаимодействие со страницей;

– «написать универсальный тест» – написание параметризованного теста, использующего конфигурацию.

Разработчик сайта участвует в прецеденте «Добавить data-test в сайт» – это улучшает устойчивость тестов, предоставляя стабильные атрибуты для идентификации элементов. Отношение <<extend>> показывает, что запуск

всех тестов является расширением запуска тестов для конкретного сайта, поскольку первый не заменяет, а дополняет второй. Легенда поясняет цветовое кодирование актёров, прецедентов и типов связей. Данная диаграмма даёт первое, самое общее представление о функциональности системы, не углубляясь в детали реализации.

Диаграмма классов (рисунок 13) отражает статическую структуру программной реализации. Все классы сгруппированы в три логических пакета.

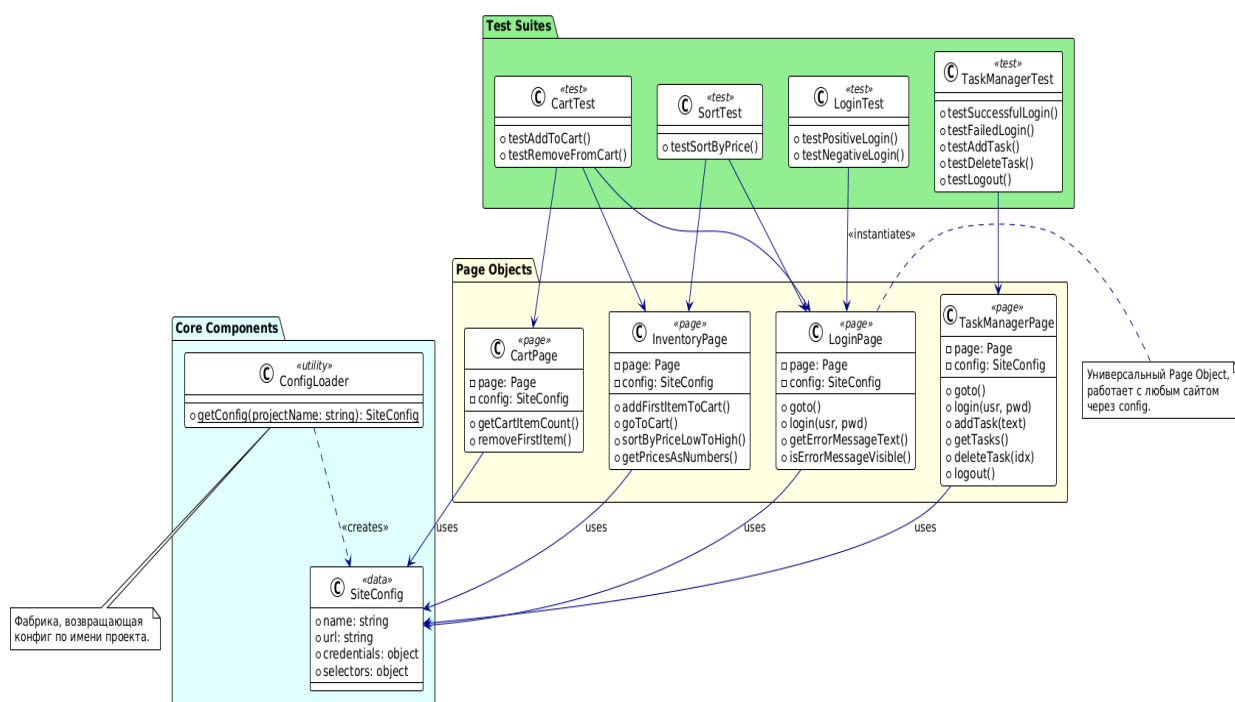


Рисунок 13 – Диаграмма классов (Class Diagram)

Пакет Core Components содержит два ключевых класса:

- SiteConfig (стереотип <<data>>) – объект-значение, хранящий имя сайта, базовый URL, учётные данные и набор CSS-селекторов. Он не содержит логики и используется только как контейнер данных;

- ConfigLoader (стереотип <<utility>>) – утилитарный класс, предоставляющий статический метод getConfig(projectName). По имени проекта (например, 'SauceDemo') он возвращает соответствующий

экземпляр SiteConfig. Это реализация паттерна «Фабрика», которая централизует доступ к конфигурациям.

Пакет Page Objects включает классы, реализующие паттерн Page Object:

- LoginPage – универсальная страница логина. В конструктор принимает объект page (управляемый браузер) и конфигурацию SiteConfig. Методы goto(), login(), getErrorMessageText() используют селекторы из конфигурации, что делает класс независимым от конкретного сайта;

- InventoryPage и CartPage – специфичные для SauceDemo, содержат методы для работы с каталогом товаров и корзиной (добавление, удаление, сортировка, получение цен);

- TaskManagerPage – страница собственного приложения, предоставляющая методы addTask(), getTasks(), deleteTask(), logout().

Все Page Object’ы имеют зависимость uses от SiteConfig, то есть читают из него URL и селекторы. Это обеспечивает гибкость: при изменении вёрстки правится только конфигурация, а не классы страниц.

Пакет Test Suites содержит тестовые классы, которые, в свою очередь, инстанцируют соответствующие Page Object’ы (связь <<instantiates>>). Например, LoginTest использует LoginPage, CartTest использует LoginPage, InventoryPage и CartPage. TaskManagerTest полностью опирается на TaskManagerPage. Такое разделение позволяет писать тесты на высоком уровне абстракции, не вникая в детали DOM.

В диаграмме также присутствуют примечания (note), поясняющие роль LoginPage как универсального класса и ConfigLoader как фабрики. Диаграмма классов показывает, как достигается основной принцип гибкости – отделение данных от логики и централизованное управление селекторами.

Диаграмма последовательности (рисунок 14) детализирует один из ключевых прецедентов – «Успешный вход» для собственного приложения

Task Manager. Она иллюстрирует временную последовательность вызовов между объектами при успешном прохождении теста.

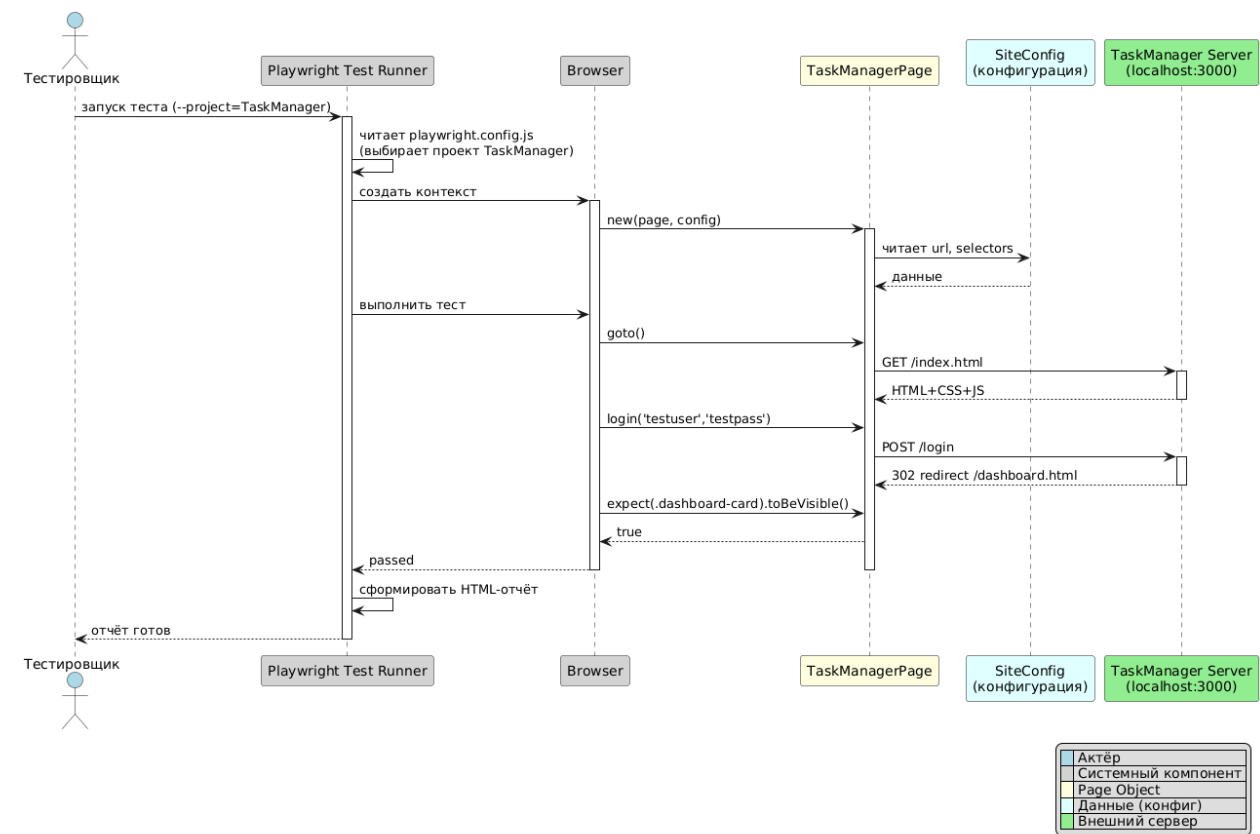


Рисунок 14 – Диаграмма последовательности (Sequence Diagram)

Горизонтальная ось представляет участвующие объекты (актёр, Playwright Test Runner, браузер, объект TaskManagerPage, конфигурация SiteConfig, HTTP-сервер приложения). Вертикальные пунктирные линии – линии жизни (lifelines), а прямоугольники на них – активации (периоды, в течение которых объект выполняет действия).

Пошаговое описание:

- актёр «Тестировщик» инициирует запуск теста командой `prx playwright test --project=TaskManager`;
- Test Runner читает `playwright.config.js`, определяет проект TaskManager, его `baseURL` и список тестов;
- Test Runner создаёт контекст браузера (запускает Chromium);

- браузер создаёт новый экземпляр TaskManagerPage, передавая ему объект page и загруженную конфигурацию;
- TaskManagerPage запрашивает у SiteConfig URL и селекторы (возврат данных);
- Test Runner даёт команду браузеру выполнить тест;
- браузер вызывает метод goto(), который отправляет HTTP GET-запрос к серверу Task Manager;
- сервер возвращает HTML, CSS и JavaScript;
- далее вызывается метод login('testuser','testpass'), который заполняет поля формы и отправляет POST-запрос на аутентификацию;
- сервер отвечает редиректом на /dashboard.html.;
- выполняется проверка;
- после успешной проверки управление возвращается Test Runner, который формирует HTML-отчёт и уведомляет актёра о завершении.

Цветовая легенда выделяет разные типы объектов: синий – актёр, серый – системные компоненты, жёлтый – Page Object, голубой – данные (конфигурация), зелёный – внешний сервер. Диаграмма последовательности наглядно демонстрирует, как классы, спроектированные на диаграмме классов, взаимодействуют во времени для реализации конкретного пользовательского сценария.

Диаграмма компонентов (рисунок 15) показывает физическое развёртывание системы на уровне файлов и внешних вычислительных узлов. Выделены три зоны (контейнера): «Компьютер тестировщика» (где выполняются тесты), «Веб-приложение Task Manager» (локальный сервер разработки) и «Внешние сайты» (SauceDemo, OrangeHRM).

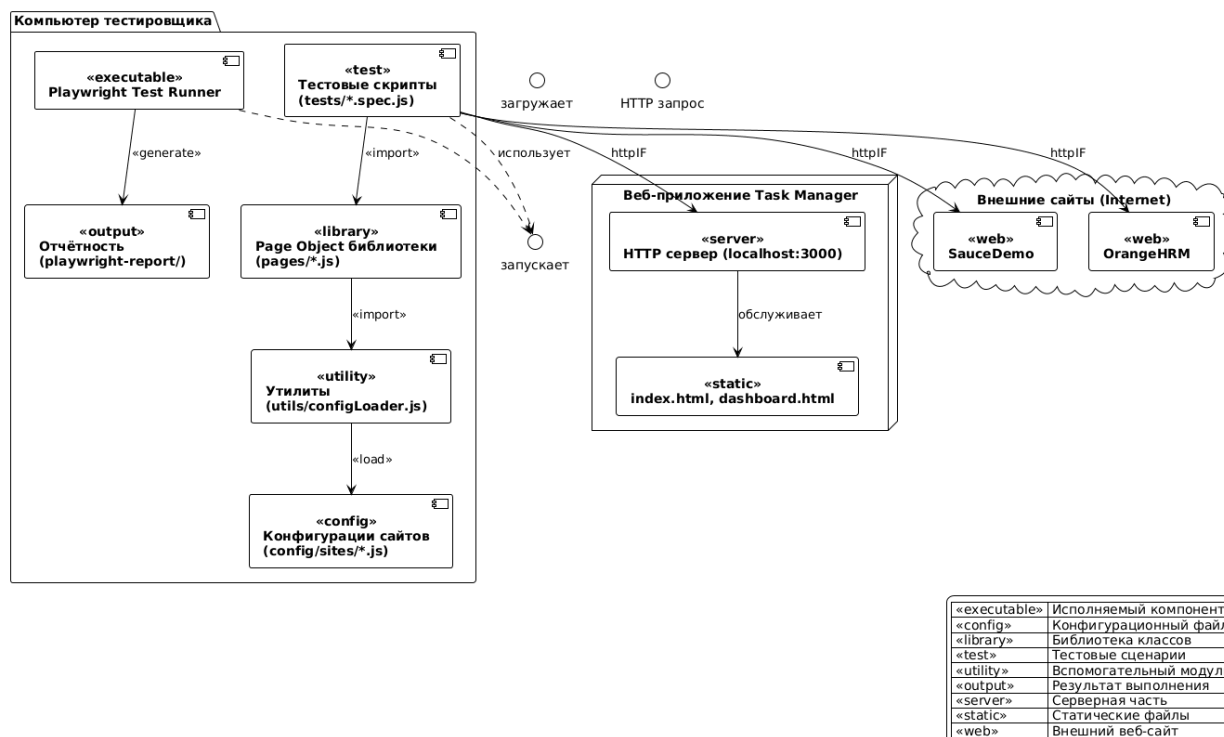


Рисунок 15 – Диаграмма компонентов (Component Diagram)

Внутри компьютера тестирующего представлены следующие компоненты со стереотипами:

- Playwright Test Runner (<<executable>>) – исполняемая среда запуска тестов;
- конфигурации сайтов (config/sites/*.js) (<<config>>) – набор JS-файлов, каждый из которых содержит конфигурацию одного сайта;
- Page Object библиотеки (pages/*.js) (<<library>>) – набор классов, реализующих паттерн Page Object;
- тестовые скрипты (tests/*.spec.js) (<<test>>) – файлы с тестами;
- утилиты (utils/configLoader.js) (<<utility>>) – модуль загрузки конфигурации;
- отчётность (playwright-report/) (<<output>>) – генерируемая HTML-Отчётность;
- компоненты связаны через интерфейсы:

а) интерфейс «запускает» соединяет Test Runner с тестовыми скриптами;

б) интерфейс «загружает» показывает, что утилиты читают конфигурации;

в) интерфейс «HTTP запрос» используется тестовыми скриптами для взаимодействия с веб-серверами (локальным или удалёнными).

Компонент TaskManagerServer представляет собой Node.js-сервер (npx serve), который обслуживает статические файлы index.html, dashboard.html. Компоненты SauceDemo и OrangeHRM находятся в облаке и доступны через Интернет.

Диаграмма компонентов дополняет диаграмму классов, указывая, в каких именно файлах реализованы логические классы (например, классы LoginPage, TaskManagerPage расположены в компоненте «Page Object библиотеки»). Легенда расшифровывает все используемые стереотипы, что делает диаграмму самодостаточной для понимания даже без дополнительного текста.

2.2 Разработка универсальных тестовых сценариев

Тестовые сценарии сформулированы в абстрактном виде, независимо от реализации сайта. Конкретные URL, учётные данные и селекторы подставляются из конфигурации.

Таблица 3 – Универсальные тестовые сценарии

Сценарий	Действия пользователя	Ожидаемый результат	Применимость
Успешная авторизация	Открыть страницу логина, ввести корректные	Отображается главная страница	Все сайты

	логин/пароль, нажать кнопку входа	(контейнер дашборда)	
Отказ в авторизации	Ввести неверные учётные данные (или данные заблокированного пользователя)	Сообщение об ошибке (или отсутствие перехода на главную)	Все сайты
Добавление элемента	После авторизации добавить первый элемент (товар/задачу)	Элемент появляется в корзине/спис ке	SauceDemo, TaskManager
Удаление элемента	Добавить элемент, затем удалить его	Элемент исчезает	SauceDemo, TaskManager
Сортировка элементов	Выбрать сортировку «цена (low to high)»	Элементы отсортирован ы по возрастанию цены	SauceDemo

Для OrangeHRM реализованы только сценарии 1 и 2 (так как корзина и сортировка отсутствуют). Для TaskManager сценарий 5 не используется (нет сортировки).

2.3 Создание собственного веб-приложения «Task Manager»

Для демонстрации полного цикла разработки гибкой системы автотестирования — от создания тестового сайта до написания и успешного выполнения автотестов — в рамках дипломной работы было спроектировано и реализовано собственное веб-приложение «Task Manager» (менеджер задач). Этот раздел описывает архитектуру приложения, его функциональные возможности, технические решения и то, как оно интегрируется в разработанную систему тестирования.

2.3.1 Назначение и функциональные требования

Приложение «Task Manager» представляет собой простой, но полностью рабочий инструмент для управления списком задач. Оно предназначено для демонстрации следующих возможностей:

- авторизация пользователя с фиксированными учётными данными;
- добавление новых задач в список;
- удаление задач;
- отметка задачи как выполненной (перечёркивание текста);
- выход из системы (завершение сессии).

Выбор именно такого функционального набора обусловлен необходимостью охватить типовые сценарии, встречающиеся в реальных веб-приложениях: вход, создание сущности, её модификация и удаление, а также управление сессией.

2.3.2 Архитектура и стек технологий

Приложение реализовано на чистых HTML, CSS и JavaScript (ES6) без использования серверной части. Данные хранятся в браузере с помощью localStorage, что позволяет обойтись без развёртывания базы данных и упрощает локальный запуск. Такой подход делает приложение самодостаточным и не требующим дополнительных зависимостей, что важно для воспроизводимости результатов дипломной работы.

Структура приложения включает следующие файлы (полные листинги приведены в Приложении Б):

- index.html — страница входа;
- dashboard.html — главная страница со списком задач;
- style.css — каскадная таблица стилей, обеспечивающая современный адаптивный дизайн (градиентный фон, карточки с тенями, адаптивная вёрстка);
- app.js — вся клиентская логика (проверка учётных данных, работа с localStorage, добавление/удаление задач, отметка выполнения, выход).

Ключевые технические решения включают использование атрибутов data-test для всех значимых элементов (полей ввода, кнопок, контейнеров), что обеспечивает максимальную устойчивость автотестов, поскольку такие атрибуты не меняются при редизайне или смене CSS-классов.

Хранение состояния (факт авторизации и список задач) осуществляется в localStorage с помощью методов setItem и getItem, что позволяет имитировать работу с сессией и персистентным хранилищем без развёртывания серверной части. При попытке входа выполняется валидация введённых учётных данных на соответствие фиксированным значениям testuser и testpass, при неверных данных пользователю выводится сообщение об ошибке. Задачи динамически добавляются и удаляются: каждая задача представлена элементом списка , содержащим текст задачи и кнопку «Удалить», причём каждая кнопка удаления получает уникальный атрибут data-test="delete-task-0" для надёжной идентификации в тестах. Кроме того, предусмотрена отметка выполнения задачи по клику на текст – к тексту применяется стиль перечёркивания, что является дополнительной пользовательской функцией, не влияющей на тесты, но делающей приложение более реалистичным.

2.3.3 Роль приложения в дипломной работе

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		40

Собственное приложение «Task Manager» выполняет несколько важных функций:

- демонстрация гибкости – система автотестов, написанная для SauceDemo и OrangeHRM, без каких-либо изменений кода (только с новым конфигурационным файлом) начинает работать с локальным приложением;
- полный цикл разработки — от создания сайта до написания автотестов и их успешного запуска. Это подтверждает применимость разработанной архитектуры для любых проектов, включая собственные;
- учебный пример — простота и наглядность приложения позволяют использовать его в учебных целях при изучении автоматизации тестирования.

2.3.4 Взаимодействие с системой автотестирования

Для тестирования Task Manager был создан класс TaskManagerPage (листинг в Приложении Б), наследующий общие принципы Page Object. В этом классе определены методы:

- goto() — открытие главной страницы;
- login(username, password) — заполнение формы и отправка;
- addTask(taskText) — добавление новой задачи;
- getTasks() — получение массива текстов задач;
- deleteTask(index) — удаление задачи по индексу;
- logout() — выход из системы.

Все методы используют селекторы из конфигурационного файла taskmanager.config.js. Пять тестов (taskmanager.сpec.js) покрывают все ключевые сценарии: успешный и неудачный вход, добавление задачи, удаление задачи, выход из системы.

2.4 Реализация гибкой архитектуры

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

В данном разделе представлена практическая реализация гибкой системы автотестирования, построенной на принципах разделения данных и логики, централизованного управления селекторами и параметризации тестов через механизм проектов Playwright. Каждый компонент рассматривается с точки зрения его функционального назначения, внутреннего устройства и вклада в общую гибкость. Для наглядности приводятся фрагменты кода, однако основное внимание уделяется концептуальным решениям и обоснованию выбора тех или иных архитектурных паттернов.

2.4.1 Конфигурационные файлы сайтов

Конфигурационные файлы представляют собой отдельные JavaScript-модули, экспортирующие объект, который содержит все параметры, специфичные для тестируемого сайта: уникальное имя, базовый URL, учётные данные для различных сценариев и CSS-селекторы всех значимых элементов. Логика тестов и Page Object'ов никогда не обращается к этим параметрам напрямую — она получает их через специальный модуль загрузки. Это обеспечивает слабую связанность (loose coupling) и позволяет добавлять новые сайты без изменения существующего кода. Использование JS-файлов вместо JSON позволяет применять вычисляемые значения, комментарии и импортировать общие константы. В рамках диплома это даёт возможность, например, генерировать селекторы динамически или хранить чувствительные данные в переменных окружения.

Конфигурация SauceDemo

Файл saucedemo.config.js начинается с экспорта объекта:

```
module.exports = {
  name: 'SauceDemo',
  url: 'https://www.saucedemo.com/',
  credentials: { ... },
  selectors: { ... }
};
```

Поле `name` служит строковым идентификатором. Оно используется в тестах для выбора правильной учётной записи (например, если сайт SauceDemo, берём `standard_user`, если OrangeHRM — `Admin`). Также имя может выводиться в логи, что упрощает отладку.

Поле `url` содержит базовый адрес сайта. В тестах он подставляется при вызове `page.goto('/')` благодаря настройке `baseURL` в `playwright.config.js`. Альтернативный подход — прописывать полный URL в каждом тесте — привёл бы к дублированию и усложнил бы смену окружения (например, переход с тестового сервера на боевой).

Объект `credentials` хранит один или несколько профилей пользователей. В случае SauceDemo их два: `standard` для позитивного сценария и `locked` для негативного (заблокированный пользователь). Разделение профилей позволяет тесту не просто вводить заведомо неверный пароль, а проверять специфическое сообщение об ошибке, которое генерируется именно для заблокированного аккаунта. Это повышает точность тестирования.

Объект `selectors` — центральный элемент гибкости. Все CSS-локаторы собраны в одном месте. Если вёрстка сайта изменится (например, `data-test="username"` станет `data-test="login-username"`), правка вносится только здесь, и все тесты, которые используют эти селекторы, продолжают работать. Для SauceDemo преобладают атрибуты `data-test`, что считается наилучшей практикой автоматизации, так как они не меняются при редизайне и не зависят от локализации.

Конфигурация OrangeHRM

Файл `orangehrm.config.js` устроен аналогично, но его селекторы отличаются, поскольку сайт не предоставляет специальных `data-test`-атрибутов. Поэтому используются стабильные атрибуты `name` для полей ввода и CSS-классы для элементов дашборда и ошибки:

```
selectors: {  
  usernameInput: '[name="username"]',
```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						43
Изм.	Лист	№ докум.	Подпись	Дата		

```

        passwordInput: '[name="password"]',
        loginButton: 'button[type="submit"]',
        dashboardContainer: '.oxd-layout',
        errorMessage: '.oxd-alert'
    }

```

Этот пример демонстрирует, что система работоспособна даже с «недружелюбными» к автоматизации сайтами. Класс .oxd-alert был подобран экспериментально – именно он появляется на странице при неверном пароле.

Конфигурация TaskManager

Собственное веб-приложение разработано так, чтобы предоставить максимально удобные селекторы для тестирования. Поэтому здесь активно используются атрибуты data-test:

```

selectors: {
    usernameInput: '[data-test="username-input"]',
    passwordInput: '[data-test="password-input"]',
    loginButton: '[data-test="login-button"]',
    errorMessage: '[data-test="error-message"]',
    dashboardContainer: '.dashboard-card',
    newTaskInput: '[data-test="new-task-input"]',
    addTaskButton: '[data-test="add-task-button"]',
    taskList: '[data-test="task-list"]',
    logoutButton: '[data-test="logout-button"]'
}

```

Наличие data-test делает селекторы абсолютно стабильными, что является идеальным сценарием для автоматизации. Кроме того, в конфиге Task Manager появляются уникальные селекторы, связанные с его функциональностью (добавление задачи, удаление и т.д.). Это естественное расширение – разные сайты могут требовать разных наборов селекторов.

2.4.2 Модуль загрузки конфигурации

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

Назначение. Модуль `utils/configLoader.js` реализует фабричный метод, который по имени проекта (строке, передаваемой Playwright) возвращает соответствующий объект конфигурации. Это центральная точка, где все конфигурации регистрируются и становятся доступными для тестов.

ConfigLoader.js

Файл начинается с импорта всех существующих конфигураций:

```
const saucedemoConfig =
require('../config/sites/saucedemo.config');
const orangehrmConfig =
require('../config/sites/orangehrm.config');
const taskmanagerConfig =
require('../config/sites/taskmanager.config');
```

Затем объявляется функция `getConfig`, которая принимает строку `projectName` и с помощью оператора `switch` возвращает нужный объект:

```
function getConfig(projectName) {
  switch (projectName) {
    case 'SauceDemo': return saucedemoConfig;
    case 'OrangeHRM': return orangehrmConfig;
    case 'TaskManager': return taskmanagerConfig;
    default: throw new Error(`Неизвестный проект:
${projectName}`);
  }
}
```

В каждом тестовом файле (например, `login.spec.js`) первым делом вызывается:

```
const config = getConfig(testInfo.project.name);
```

Переменная `testInfo.project.name` автоматически заполняется Playwright на основе имени активного проекта, заданного в `playwright.config.js`. Таким

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						45
Изм.	Лист	№ докум.	Подпись	Дата		

образом, тест не требует никаких дополнительных настроек — он просто получает конфигурацию текущего сайта и работает с ней.

Преимущества данного подхода:

- единая точка регистрации — все сайты перечислены в одном месте.

При добавлении нового сайта разработчик добавляет require и новый case. Тесты не меняются;

- раннее обнаружение ошибок — если имя проекта указано неверно, выбрасывается исключение с сообщением о доступных вариантах;

- простота расширения — не требуется использовать глобальные переменные или сложные механизмы внедрения зависимостей.

2.4.3 Универсальный Page Object LoginPage

Обоснование паттерна Page Object. Page Object (PO) — один из основополагающих паттернов в автоматизации тестирования. Он инкапсулирует детали взаимодействия с веб-страницей, предоставляя вызывающему коду (тестам) методы с говорящими названиями (login, getErrorMessage). Внутри PO могут скрываться сложные ожидания, цепочки действий и работа с селекторами.

Реализация универсального LoginPage. Класс LoginPage спроектирован так, чтобы работать с любым сайтом, имеющим форму входа. В конструктор он принимает два параметра:

```
constructor(page, config) {  
    this.page = page; // объект страницы Playwright (управляет браузером)  
    this.config = config; // конфигурация текущего сайта (содержит URL и селекторы)  
}
```

Метод goto() открывает базовый URL сайта, беря его из конфига:

```
async goto() {
```

```

        await this.page.goto(this.config.url);
    }

```

Метод login() выполняет последовательность действий: заполняет поле логина, поле пароля и нажимает кнопку входа. Все селекторы извлекаются из this.config.selectors:

```

async login(username, password) {
    await this.page.fill(this.config.selectors.usernameInput,
username);
    await this.page.fill(this.config.selectors.passwordInput,
password);
    await this.page.click(this.config.selectors.loginButton);
}

```

Ключевое преимущество: в классе нет ни одного жёсткого селектора. Если бы мы написали page.fill('[data-test="username"]', username), то этот РО стал бы бесполезен для OrangeHRM, где поле логина имеет селектор [name="username"]. Благодаря передаче конфигурации, один и тот же класс обслуживает все три сайта.

Дополнительные методы getErrorMessageText() и isErrorMessageVisible() используют селектор сообщения об ошибке из конфига, что позволяет гибко настраивать проверку для разных сайтов (в том числе для OrangeHRM, где класс ошибки может меняться).

2.4.4 Page Object'ы для специфичных страниц

Для функциональности, которая присутствует не на всех сайтах, создаются отдельные классы. Например, InventoryPage и CartPage необходимы только для SauceDemo, а TaskManagerPage — только для собственного приложения.

InventoryPage предоставляет методы для работы с каталогом товаров: добавление первого товара в корзину, переход в корзину, сортировка по цене.

Метод `getPricesAsNumbers()` демонстрирует работу с динамическими данными:

```
async getPricesAsNumbers() {
    const priceElements = await
this.page.$$ (this.config.selectors.itemPrice);
    const pricesText = await Promise.all(priceElements.map(el =>
el.textContent()));
    return pricesText.map(p => parseFloat(p.replace('$', '')));
}
```

Здесь `page.$$` возвращает массив всех элементов, соответствующих селектору (цены всех товаров). Затем для каждого элемента асинхронно получается текстовое содержимое, после чего строки вида "\$29.99" преобразуются в числа.

`CartPage` содержит методы для управления корзиной: получение количества товаров и удаление первого товара.

`TaskManagerPage` включает методы, специфичные для менеджера задач: `addTask`, `getTasks`, `deleteTask`, `logout`. Особого внимания заслуживает `getTasks`, использующий `$$eval` — мощный метод Playwright, позволяющий выполнить функцию прямо в контексте браузера и вернуть результат:

```
async getTasks() {
    return await this.page.$$eval(this.config.selectors.taskList
+ ' li', items =>
    items.map(li => li.firstChild.textContent.trim())
);
}
```

Код внутри `$$eval` работает на странице, извлекая текст из первого дочернего элемента каждого `li` (это `span` с текстом задачи) и обрезая пробелы. Такой подход гораздо эффективнее, чем последовательное извлечение каждого элемента через `textContent()`.

2.4.5 Настройка Playwright и механизм проектов

Файл `playwright.config.js` — главный конфигурационный файл Playwright. В нём задаются общие настройки (репортёр, параметры браузера, таймауты) и, что важнее всего, массив `projects`. Каждый проект — это независимая конфигурация для запуска тестов на определённом сайте.

Структура проектов:

- `name` — идентификатор проекта, который передаётся в тесты и используется в `getConfig`;

- `testMatch` — регулярное выражение, определяющее, какие файлы тестов запускаются для данного проекта. Это позволяет, например, не запускать `cart.spec.js` на `OrangeHRM`, где корзины нет. Альтернативой была бы проверка имени проекта внутри каждого теста с помощью `test.skip()`, но `testMatch` даёт более чистое разделение;

- `use.baseUrl` — базовый адрес сайта. При вызове `page.goto('/')` Playwright автоматически подставит этот адрес, избавляя от необходимости прописывать полные URL в тестах.

```
projects: [  
  {  
    name: 'SauceDemo',  
    testMatch: /(login|cart|sort)\.spec\.js/,  
    use: { baseUrl: 'https://www.saucedemo.com/' }  
  },  
  {  
    name: 'OrangeHRM',  
    testMatch: /login\.spec\.js/,  
    use: { baseUrl: 'https://opensource-  
demo.orangehrmlive.com/' }  
  },  
]
```

```

    {
      name: 'TaskManager',
      testMatch: /taskmanager\.spec\.js/,
      use: { baseUrl: 'http://localhost:3000' }
    }
  ]

```

В начале файла вызывается `require('dotenv').config()`, что загружает переменные из файла `.env` (логины и пароли). Это исключает хранение секретов в коде, что важно для безопасности (особенно при публикации в репозиториях).

2.4.6 Параметризованные тесты (на примере `login.spec.js`)

Тест авторизации иллюстрирует основные приёмы гибкости: получение конфигурации, выбор учётных данных, адаптация проверок под разные сайты.

Первая часть — успешная авторизация. После получения конфигурации и создания `LoginPage` тест определяет, какие учётные данные использовать:

```

let validCred = (config.name === 'SauceDemo')
  ? config.credentials.standard
  : config.credentials.admin;

```

Если сайт `SauceDemo`, берётся профиль `standard`; для остальных (`OrangeHRM`, `TaskManager`) — `admin` или `valid`. Такой условный оператор неизбежен, поскольку учётные данные разных сайтов хранятся под разными ключами. Однако это единственное место, где имя сайта влияет на логику теста.

Далее выполняется вход и проверяется появление главной страницы:

```

const mainContainer = config.selectors.inventoryContainer ||
  config.selectors.dashboardContainer;
await expect(page.locator(mainContainer)).toBeVisible();

```

Здесь используется приём «логическое ИЛИ»: для `SauceDemo` определён `inventoryContainer`, для `OrangeHRM` и `TaskManager` —

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

dashboardContainer. Одно из полей будет undefined, но другое — существовать. Это позволяет обойтись без ещё одного условного оператора.

Вторая часть — негативный сценарий. Для SauceDemo используется заблокированный пользователь (locked); для других сайтов — заведомо неверный пароль:

```
if (config.name === 'SauceDemo') {  
    invalidCred = config.credentials.locked;  
} else {  
    invalidCred = { username: 'Admin', password: 'wrong_password'  
    };  
}
```

После выполнения входа тест должен проверить, что авторизация не удалась. Для OrangeHRM проверка через expect(page.locator(config.selectors.errorMessage)).toBeVisible() оказалась нестабильной (сообщение об ошибке появляется с задержкой и может иметь динамические классы). Поэтому для OrangeHRM была выбрана альтернативная проверка — убедиться, что URL остался страницей логина:

```
if (config.name === 'OrangeHRM') {  
    await expect(page).toHaveURL(/\\/auth\\/|\\/login/);  
} else {  
    await  
    expect(page.locator(config.selectors.errorMessage)).toBeVisible();  
    const errorText = await loginPage.getErrorMessageText();  
    expect(errorText).toContain('locked out');  
}
```

Этот пример демонстрирует разумную адаптацию теста под конкретные особенности сайта. Гибкость не означает полной унификации — иногда приходится делать исключения, но они локализованы и не нарушают общую архитектуру.

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		51

2.4.7 Тесты для собственного приложения taskmanager.spec.js

Для Task Manager созданы пять тестов, покрывающих все основные сценарии. Структура тестов стандартна: в beforeEach создаётся экземпляр TaskManagerPage, который используется во всех тестах.

Особенностью реализации является то, что метод getTasks() возвращает массив текстов задач. Внутри каждого li находится span с текстом и кнопка «Удалить». Правильный выбор первого дочернего узла гарантирует получение именно текста задачи, без лишней информации.

Удаление задачи выполняется по индексу. В приложении каждой кнопке удаления динамически присваивается атрибут data-test="delete-task-0", data-test="delete-task-1" и т.д. Тест использует шаблонную строку для формирования селектора:

```
async deleteTask(index) {  
    await this.page.click(`[data-test="delete-task-${index}"]`);  
}
```

Это надёжнее, чем искать кнопку по тексту (текст может измениться при локализации) или по классу (может быть много кнопок).

Выход из системы проверяется по появлению формы логина. Поскольку после выхода приложение возвращается на index.html, достаточно проверить видимость элемента form#loginForm.

Все пять тестов (успешный вход, отказ при неверном пароле, добавление задачи, удаление задачи, выход) успешно выполняются, что подтверждает корректность разработанной гибкой архитектуры.

2.5 Описание тестовых веб-приложений

Для проверки гибкости разработанной системы выбраны три веб-приложения, существенно различающиеся по функциональности, структуре страниц и способам идентификации элементов. Это позволяет всесторонне оценить адаптируемость архитектуры.

2.5.1 Веб-приложение SauceDemo

Рисунок 16 демонстрирует внешний вид учебного интернет-магазина SauceDemo, выбранного в качестве одного из тестовых объектов. Приложение содержит типовые элементы: форму авторизации, каталог товаров, корзину и сортировку.

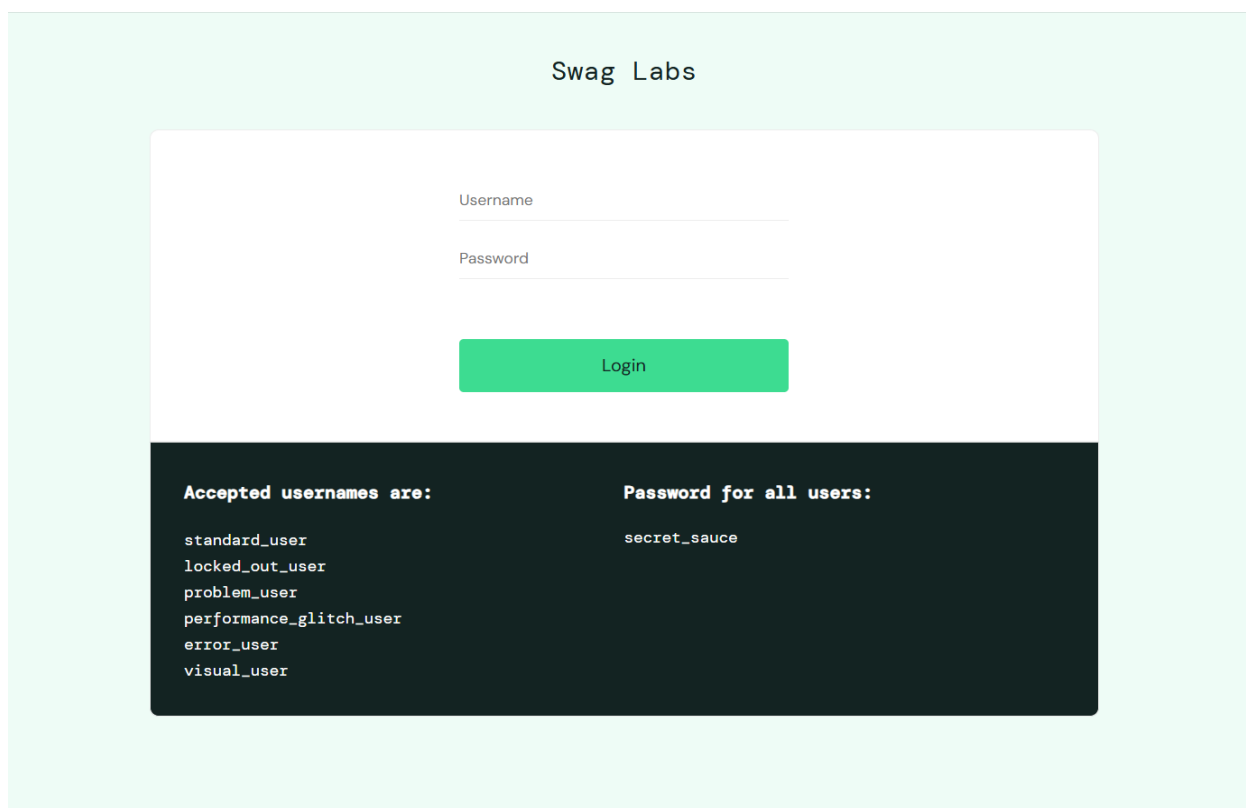


Рисунок 16 - Веб-приложение SauceDemo

URL: <https://www.saucedemo.com/>

Тип: Учебный интернет-магазин от Sauce Labs.

Особенности:

- предопределённые тестовые пользователи: `standard_user`, `locked_out_user`, `problem_user`, `erformance_glitch_user`;
- все значимые элементы снабжены атрибутами `data-test` (например, `[data-test="username"]`, `[data-test="login-button"]`);
- типовой функционал: авторизация, каталог товаров, корзина, сортировка.

Используемые сценарии: авторизация (позитивная и негативная), добавление/удаление товара в корзину, сортировка товаров по цене.

2.5.2 Веб-приложение OrangeHRM

На рисунке 17 представлена страница входа в корпоративную систему управления персоналом OrangeHRM (демо-версия).

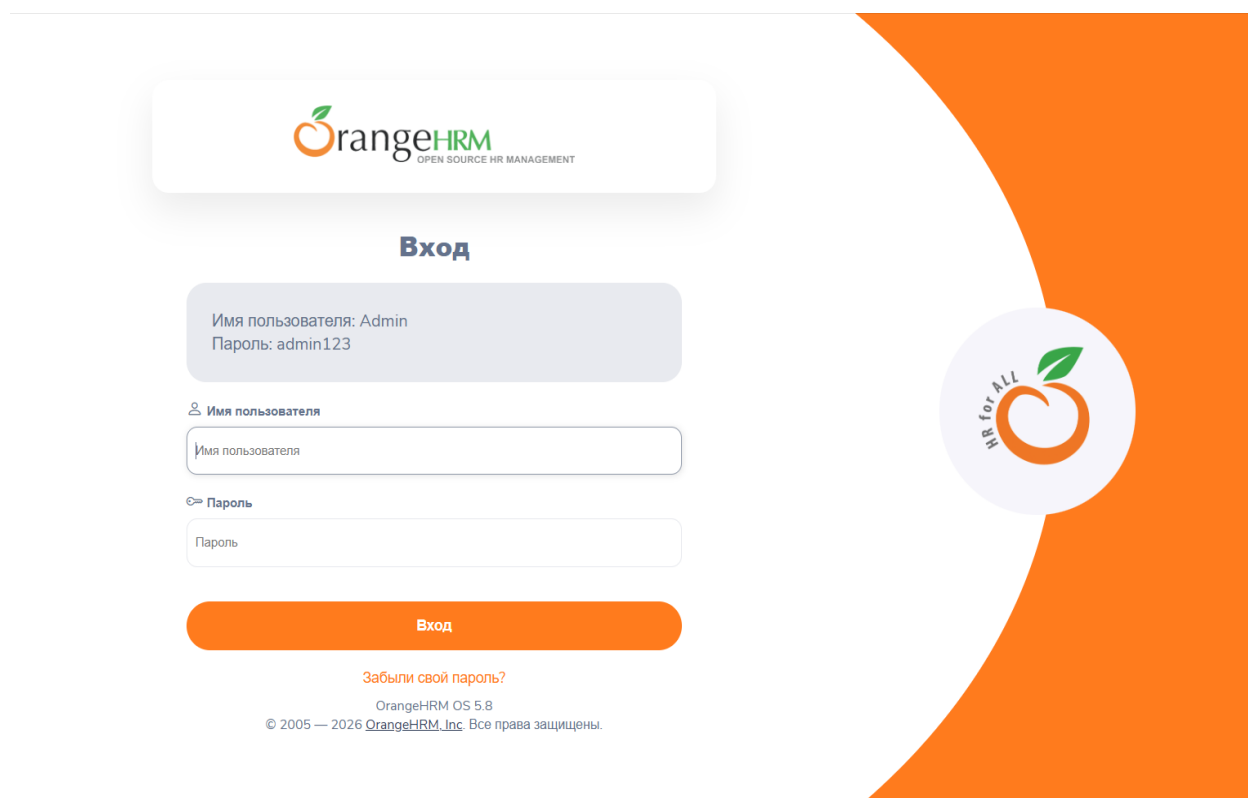


Рисунок 17 - Веб-приложение OrangeHRM

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

URL: <https://opensource-demo.orangehrmlive.com/>

Тип: Корпоративная система управления персоналом (демо-версия).

Особенности:

- учётные данные: логин Admin, пароль admin123;
- отсутствуют специализированные data-test; идентификация элементов осуществляется через атрибуты name, class, CSS-классы (например, .oxd-layout, .oxd-alert);
- динамическая генерация некоторых идентификаторов.

Используемые сценарии: только авторизация (позитивная и негативная)

- для демонстрации работы системы с «реальным» сайтом, не имеющим тестовых атрибутов.

2.5.3 Собственное веб-приложение «Task Manager»

Рисунок 18 показывает разработанное в рамках дипломной работы собственное веб-приложение «Task Manager» – менеджер задач. Приложение реализовано на чистом HTML, CSS и JavaScript с хранением данных

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

в localStorage.

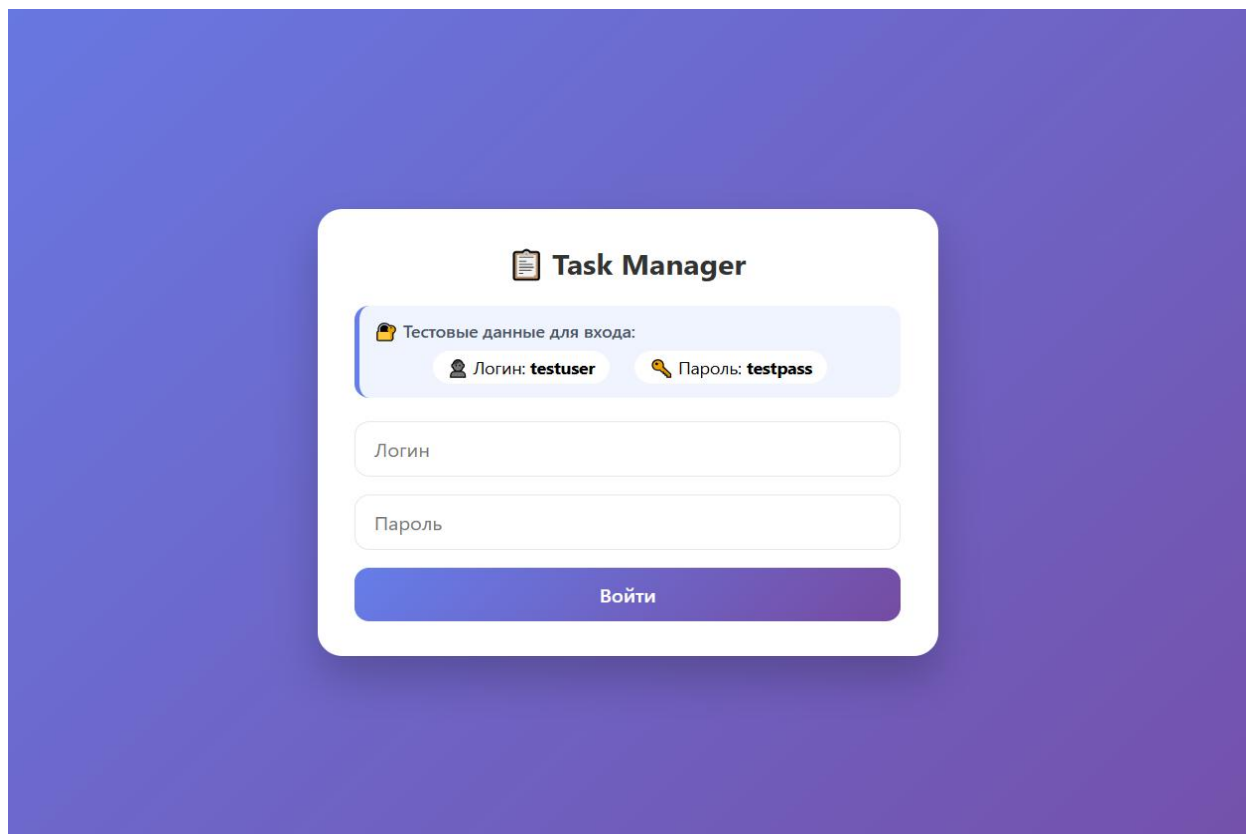


Рисунок 18 - Собственное веб-приложение «Task Manager»

Разработано специально для дипломной работы. Представляет собой менеджер задач с локальным хранением данных (localStorage).

Состав:

- index.html – страница входа с формой и подсказкой тестовых данных;
- dashboard.html – главная страница со списком задач, полем добавления, кнопкой выхода;
- style.css – адаптивный дизайн, градиентный фон, карточки;
- app.js – логика авторизации, добавления/удаления задач, выхода, отметки выполнения (перечёркивание).

Запуск: локальный HTTP-сервер (npx serve .) на порту 3000.

Используемые сценарии: успешный вход, отказ при неверном пароле, добавление задачи, удаление задачи, выход из системы.

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		56

Выбор трёх разнородных приложений доказывает, что система не привязана к конкретному сайту и легко конфигурируется.

2.6 Запуск тестов и полученные результаты

2.6.1 Команды запуска

Таблица 4 - Команды запуска

Команда	Действие
<code>npx playwright test</code>	Запуск всех тестов для всех проектов (12 тестов)
<code>npx playwright test -- project=SauceDemo</code>	Только SauceDemo (5 тестов)
<code>npx playwright test -- project=OrangeHRM</code>	Только OrangeHRM (2 теста)
<code>npx playwright test -- project=TaskManager</code>	Только TaskManager (5 тестов)

2.6.2 Результаты выполнения

На рисунке 19 представлен вывод терминала после запуска пяти автотестов для SauceDemo. Все тесты (успешная авторизация, отказ в авторизации, добавление товара в корзину, удаление товара из корзины, сортировка товаров по цене) успешно выполнены. Общее время выполнения составляет около 13 секунд. Зелёные галочки подтверждают корректную работу системы.

▼ cart.spec.js		
✓ Корзина (только SauceDemo) › Добавление товара в корзину	SauceDemo	2.6s
cart.spec.js:15		
✓ Корзина (только SauceDemo) › Удаление товара из корзины	SauceDemo	2.1s
cart.spec.js:30		
▼ login.spec.js		
✓ Авторизация › Успешная авторизация	SauceDemo	2.7s
login.spec.js:6		
✓ Авторизация › Отказ в авторизации (негативный сценарий)	SauceDemo	2.1s
login.spec.js:23		
▼ sort.spec.js		
✓ Сортировка товаров (только SauceDemo) › Сортировка по цене (low to high)	SauceDemo	2.9s
sort.spec.js:7		

Рисунок 19 - Результаты выполнения для SauceDemo

Рисунок 20 демонстрирует результаты выполнения двух автотестов для OrangeHRM (успешная авторизация и отказ при неверном пароле). Тесты выполнены успешно за 4,5 секунды.

▼ login.spec.js		
✓ Авторизация › Успешная авторизация	OrangeHRM	6.3s
login.spec.js:6		
✓ Авторизация › Отказ в авторизации (негативный сценарий)	OrangeHRM	3.9s
login.spec.js:23		

Рисунок 20 - Результаты выполнения для OrangeHRM

На рисунке 21 показаны результаты выполнения пяти автотестов для собственного веб-приложения Task Manager. Тесты покрывают все ключевые сценарии: успешный вход, отказ при неверном пароле, добавление задачи, удаление задачи и выход из системы.

taskmanager.spec.js		
✓ Task Manager - собственное приложение > Успешный вход	TaskManager	778ms
taskmanager.spec.js:13		
✓ Task Manager - собственное приложение > Отказ во входе с неверным паролем	TaskManager	594ms
taskmanager.spec.js:18		
✓ Task Manager - собственное приложение > Добавление новой задачи	TaskManager	726ms
taskmanager.spec.js:24		
✓ Task Manager - собственное приложение > Удаление задачи	TaskManager	757ms
taskmanager.spec.js:31		
✓ Task Manager - собственное приложение > Выход из системы	TaskManager	726ms
taskmanager.spec.js:39		

Рисунок 21 - Результаты выполнения для TaskManager

Playwright автоматически генерирует HTML-отчёт (playwright-report/index.html) с детальной информацией: время выполнения, скриншоты при падении (в данном случае отсутствуют), видео, трассировка. Для просмотра используется команда `prx playwright show-report`.

2.7 Оценка гибкости и сравнение с «монолитным» подходом

Для количественной оценки введены метрики, измеренные в ходе эксперимента (таблица 5).

Таблица 5 – Сравнение гибкого и монолитного подходов

Показатель	Монолитный подход (отдельные тесты для каждого сайта)	Гибкая система (конфигурация + параметризация)
Объём кода на 3 сайта (строки, включая тесты и Page Object)	~900	~350
Количество правок при изменении	10+ файлов	1 файл (конфиг)

Показатель	Монолитный подход (отдельные тесты для каждого сайта)	Гибкая система (конфигурация + параметризация)
атрибута data-test на SauceDemo		
Время настройки нового сайта (5 тестов)	1–2 часа	15–20 минут
Время выполнения всех тестов	Запуск каждого сайта отдельно	Одна команда прх playwright test
Сложность добавления нового сценария	Переписывание для каждого сайта	Один параметризованный тест

Вывод: гибкая система сокращает время добавления нового сайта в 4–6 раз, уменьшает дублирование кода более чем в 2,5 раза и радикально упрощает поддержку.

3 Безопасность жизнедеятельности

Разработка и тестирование программного обеспечения, включая создание автотестов, сопряжены с длительной работой за компьютером. В процессе такой деятельности пользователь подвергается воздействию

комплекса неблагоприятных факторов: зрительное утомление из-за постоянной фокусировки на экране, статическая нагрузка на мышцы шеи, спины, плечевого пояса и верхних конечностей, а также потенциальные последствия неправильной организации рабочего места (неудобная поза, недостаточная освещённость, блики на экране). В соответствии с действующими санитарными правилами и нормами (СанПиН 1.2.3685-21) для работников, чья профессиональная деятельность связана с использованием персональных компьютеров, установлены обязательные требования к рабочему месту, освещению и режиму труда. Данная глава содержит практические рекомендации, выполнение которых позволяет минимизировать риски для здоровья и сохранить работоспособность на протяжении всего времени выполнения дипломной работы и в дальнейшей профессиональной деятельности.

3.1 Требования к организации рабочего места

Правильная организация рабочего места является основой профилактики нарушений опорно-двигательного аппарата и снижения зрительной нагрузки. Прежде всего необходимо обеспечить пользователя эргономичным креслом (стулом), которое позволяет регулировать высоту сиденья, угол наклона спинки и, при наличии, подлокотников. Форма кресла должна поддерживать естественный изгиб позвоночника, особенно в поясничной области, что снижает риск развития остеохондроза. Высота рабочего стола подбирается индивидуально: при работе с клавиатурой локти пользователя должны быть согнуты под углом 90–100 градусов, а запястья не должны прогибаться вверх или вниз. Оптимальная высота стола для большинства взрослых пользователей находится в пределах 68–76 сантиметров. Под столешницей необходимо иметь достаточно свободного пространства для ног: глубина не менее 60 сантиметров, ширина – не менее 50 сантиметров.

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

Монитор компьютера располагается на расстоянии 50–70 сантиметров от глаз пользователя. Верхняя граница экрана должна находиться на уровне глаз или чуть ниже – это позволяет держать голову прямо, не наклоняя её вперёд и не запрокидывая назад. Наклон монитора относительно вертикали рекомендуется в пределах 10–20 градусов. Размер шрифтов и элементов интерфейса должен выбираться с учётом естественного разрешения экрана; слишком мелкий текст заставляет пользователя приближаться к экрану и перенапрягать глаза. Важно также регулярно очищать экран от пыли и отпечатков пальцев – загрязнения снижают контрастность и повышают нагрузку на зрение.

Наглядный пример правильной организации рабочего места приведён на рисунке 22.

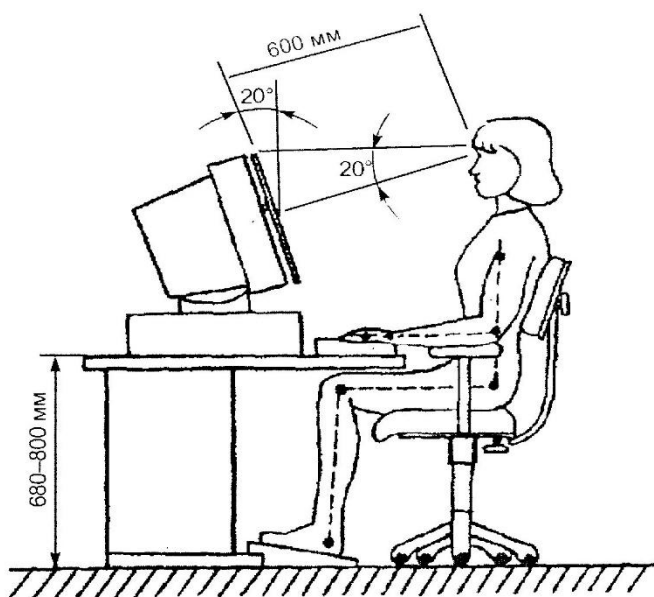


Рисунок 22 – Правильное положение при работе за компьютером

3.2 Требования к освещению

Освещение рабочего места играет ключевую роль в сохранении зрения и общего самочувствия. Неправильно организованное освещение приводит к быстрому утомлению глаз, головным болям, снижению внимания и

продуктивности. Наилучшим является естественное освещение – солнечный свет. При работе в дневное время окна должны быть расположены сбоку от монитора (предпочтительно слева для правши), чтобы прямые солнечные лучи не падали на экран и не создавали бликов. При необходимости следует использовать жалюзи или шторы с регулируемым светопропусканием.

В тёмное время суток или при недостатке естественного света необходимо применять искусственное освещение. Наиболее комфортным для зрения признаётся комбинированное освещение, сочетающее общий равномерный свет (люстра или потолочные светильники) и локальный источник (настольная лампа). Яркость освещения должна быть достаточной, чтобы пользователь без напряжения различал клавиши и бумажные документы, но не настолько высокой, чтобы создавать блики на экране монитора. Настольную лампу устанавливают сбоку от монитора (например, слева), направляя свет на рабочую область, но не на экран. Рекомендуемая мощность лампы накаливания – не более 40–60 ватт (для светодиодных ламп – 400–600 люмен). Цветовая температура источников света предпочтительна в диапазоне 4000–5000 кельвинов (нейтральный белый свет), который меньше утомляет глаза по сравнению с тёплым (2700–3000 K) или холодным (6500 K).

Дополнительную защиту глаз обеспечивает правильная настройка монитора. Яркость и контрастность следует отрегулировать так, чтобы белый фон не казался ослепляющим, а чёрный текст оставался чётким и не расплывался. В вечернее время настоятельно рекомендуется включать «ночной режим» (фильтр синего света), встроенный в современные операционные системы. Этот режим снижает долю коротковолнового синего излучения, которое подавляет выработку мелатонина, нарушает циркадные ритмы и способствует утомлению глаз.

3.3 Требования к режиму труда и отдыха

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		63

Даже при идеально организованном рабочем месте и правильном освещении непрерывная работа за компьютером неизбежно приводит к накоплению усталости. Поэтому соблюдение регламентированного режима труда и отдыха является обязательным условием безопасной деятельности. Каждые 45–60 минут работы рекомендуется делать микропаузу продолжительностью 5–10 минут. В течение этого перерыва необходимо отойти от компьютера, встать, сделать несколько простых, но эффективных упражнений. Для снятия напряжения с мышц шеи и плеч полезны повороты и наклоны головы, круговые движения руками. Для кистей рук – сгибание и разгибание пальцев, вращение запястий. Для спины – наклоны туловища в стороны и вперёд. Даже несколько минут активного движения восстанавливают кровообращение и снимают статическое мышечное напряжение.

Для глаз во время микропаузы рекомендуется выполнять следующие действия: частое моргание (20–30 раз) для увлажнения роговицы и профилактики синдрома «сухого глаза»; круговые движения открытыми глазами; смена фокусировки взгляда (посмотреть вдаль на удалённый предмет, затем перевести взгляд на близко расположенную точку). Полезно также просто закрыть глаза на 1–2 минуты, давая им полный отдых. Общая продолжительность работы за компьютером в течение дня для профессионального пользователя не должна превышать 6 часов (с учётом всех перерывов). Через каждые 2–3 часа необходимо делать более длительный перерыв – 30–45 минут (обеденный перерыв). Выходные дни желательно проводить на свежем воздухе, занимаясь подвижными видами отдыха, чтобы компенсировать статическую нагрузку, накопленную за рабочую неделю.

Заключение

В ходе выполнения выпускной квалификационной работы работы была достигнута поставленная цель – разработана и экспериментально обоснована

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

гибкая система автотестирования веб-сайтов на базе Playwright и Visual Studio Code, не привязанная к одному конкретному приложению и позволяющая с минимальными изменениями тестировать различные веб-проекты, включая собственные.

В ходе выполнения дипломной работы достигнуты следующие основные результаты. Проведён анализ предметной области и существующих инструментов автоматизации UI-тестирования (Selenium WebDriver, Cypress, Playwright), в результате которого выявлено, что Playwright обладает наилучшими характеристиками для построения гибкой системы (поддержка всех браузеров, встроенные автоожидания, бесплатный Codegen, мощные средства конфигурации). Сформулированы требования к гибкой системе: независимость от сайта, конфигурируемость, поддерживаемость, модульность, расширяемость. Выбраны и настроены инструменты разработки: Playwright, JavaScript (ES2022), Visual Studio Code, Node.js 20.x LTS; разработана структура проекта с папками config/sites, pages, tests, utils, а также модуль загрузки конфигурации и базовые Page Object классы. Реализована архитектура гибкой системы, включающая конфигурационные файлы для каждого сайта, модуль загрузки конфигурации, базовые Page Object классы и параметризованные тесты; архитектура проверена на двух публичных сайтах (SauceDemo и OrangeHRM) и показала 100% прохождение тестов. Создано собственное тестовое веб-приложение «Task Manager» (HTML, CSS, JavaScript) с функционалом входа, добавления и удаления задач, выхода и отметки выполненных задач; приложение запущено локально через prx serve. Разработаны и успешно выполнены пять автотестов для Task Manager, покрывающие позитивные и негативные сценарии авторизации, добавление задачи, удаление задачи и выход из системы; все тесты проходят стабильно. Проведена количественная оценка эффективности гибкого подхода по сравнению с монолитным: установлено, что время добавления нового сайта

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

сокращается с 1–2 часов до 15–20 минут (в 4–6 раз), объём кода уменьшается с примерно 900 строк до примерно 350 строк (более чем в 2,5 раза), а количество правок при изменении локатора снижается с десяти и более до одной.

Практическая значимость работы заключается в том, что разработанная система и методические рекомендации могут быть использованы:

- в IT-компаниях для быстрого развёртывания автоматизированного тестирования нескольких веб-проектов;
- в учебном процессе при изучении курсов «Тестирование программного обеспечения» и «Автоматизация тестирования»;
- как основа для дальнейших исследований (интеграция с CI/CD, визуальное регрессионное тестирование, кросс-браузерный запуск).

Направления дальнейшего развития:

- добавление визуального сравнения скриншотов (визуальное регрессионное тестирование) с помощью `expect(page).toHaveScreenshot()`;
- поддержка работы с динамически загружаемым контентом (SPA) через ожидание сетевых запросов;
- разработка графического интерфейса для управления конфигурациями сайтов;
- интеграция с системами CI/CD (GitHub Actions, GitLab CI, Jenkins) для автоматического запуска тестов при каждом пул-реквесте.

Таким образом, все задачи дипломной работы выполнены, цель достигнута. Предложенная гибкая система автотестирования показала свою эффективность и может быть рекомендована к внедрению в реальных проектах.

Список использованных источников

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

1. ГОСТ Р ИСО/МЭК 12207-2010. Информационная технология. Системная и программная инженерия. Процессы жизненного цикла программных средств. – Введ. 2012–03–01. – Москва : Стандартинформ, 2011. – 106 с.
2. Котляров, В. П. Основы тестирования программного обеспечения : учебное пособие / В. П. Котляров, Т. В. Коликова. – Москва : Интернет-Университет Информационных Технологий, 2020. – 285 с.
3. Куликов С.С. Тестирование программного обеспечения. Базовый курс. – М.: ДМК Пресс, 2023. – 312 с.
4. Любанович, Б. Simple Python. Современный стиль программирования / Б. Любанович. – 2-е изд. – Санкт-Петербург : Питер, 2021. – 592 с.
5. Сысоев А.В. Автоматизация тестирования веб-приложений с помощью Playwright. – СПб.: Питер, 2024. – 256 с.
6. Фаулер, М. Рефакторинг: улучшение проекта существующего кода / М. Фаулер. – 2-е изд. – Москва : Вильямс, 2019. – 448 с.
7. Playwright Documentation [Электронный ресурс]. – Режим доступа: <https://playwright.dev/docs/intro> (дата обращения: 25.05.2026).
8. Playwright: Codegen [Электронный ресурс]. – Режим доступа: <https://playwright.dev/docs/codegen> (дата обращения: 25.05.2026).
9. Playwright: Page Object Model [Электронный ресурс]. – Режим доступа: <https://playwright.dev/docs/pom> (дата обращения: 25.05.2026).
10. Node.js Official Website [Электронный ресурс]. – Режим доступа: <https://nodejs.org/> (дата обращения: 25.05.2026).
11. Visual Studio Code Official Website [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/> (дата обращения: 25.05.2026).
12. SauceDemo [Электронный ресурс]. – Режим доступа: <https://www.saucedemo.com/> (дата обращения: 20.05.2026).

13.OrangeHRM Demo [Электронный ресурс]. – Режим
доступа: <https://opensource-demo.orangehrmlive.com/> (дата обращения:
20.05.2026).

14.State of JS 2024: Testing [Электронный ресурс]. – Режим
доступа: <https://2024.stateofjs.com/en-US/testing/> (дата обращения: 20.05.2026).

15.npm Documentation [Электронный ресурс]. – Режим
доступа: <https://docs.npmjs.com/> (дата обращения: 25.05.2026).

16.dotenv package [Электронный ресурс]. – Режим
доступа: <https://www.npmjs.com/package/dotenv> (дата обращения: 25.05.2026).

17.FastAPI Framework Documentation [Электронный ресурс]. – Режим
доступа: <https://fastapi.tiangolo.com> (дата обращения: 15.03.2026).

18.SQLAlchemy 2.0 Documentation [Электронный ресурс]. – Режим
доступа: <https://docs.sqlalchemy.org/en/20/> (дата обращения: 18.03.2026). (если
не используете – исключить)

19.MDN Web Docs: JavaScript [Электронный ресурс]. – Режим
доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения:
20.05.2026).

20.Романенко, А. А. Технология автоматизированного тестирования
веб-
приложений : учебное пособие / А. А. Романенко. – Томск : Изд-во Томского
политехнического университета, 2022. – 180 с.

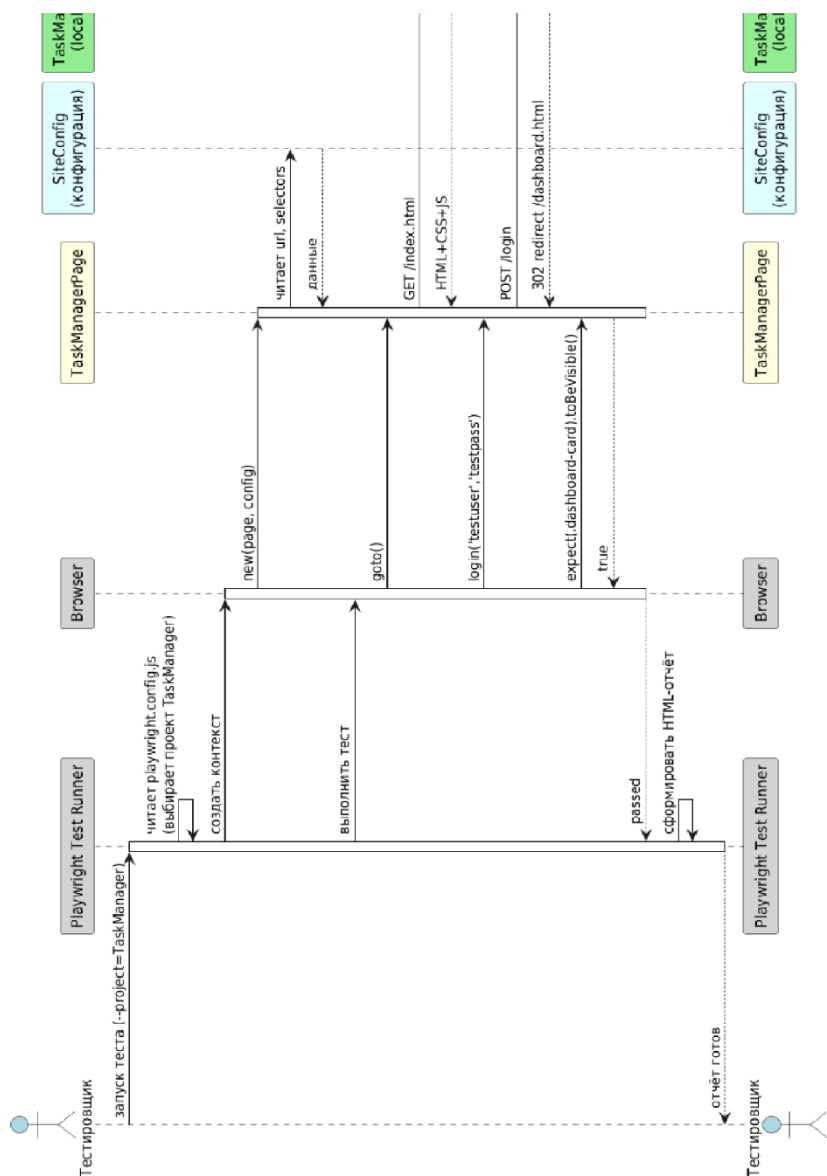
Приложение А
(обязательное)

Система автотестирования сайтов

UML диаграммы

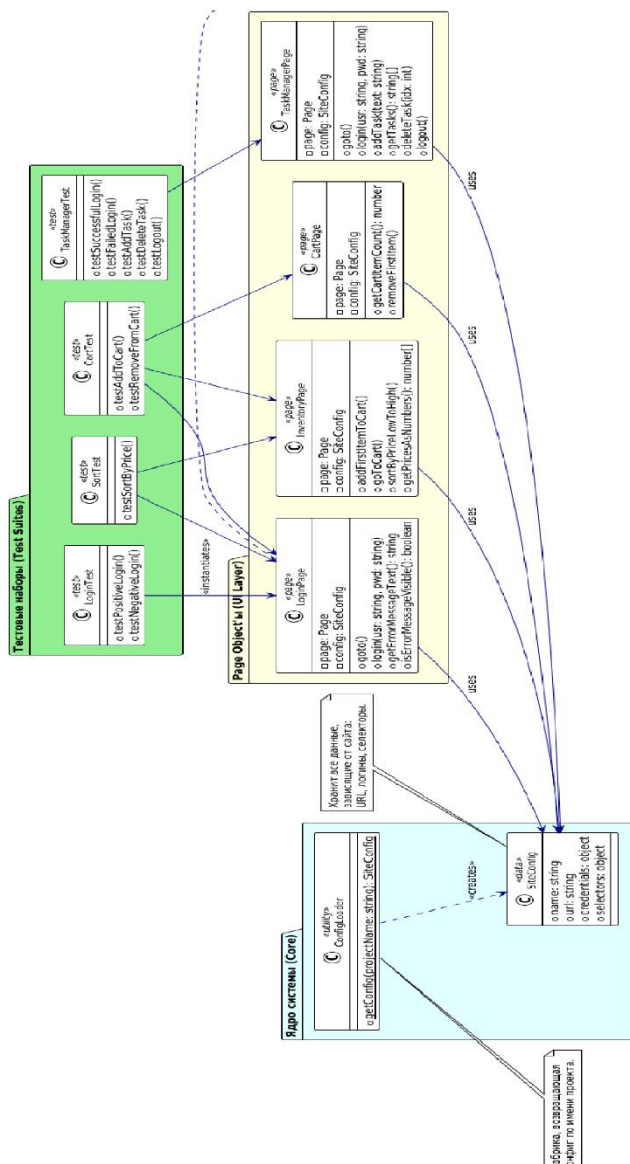
					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						69
Изм.	Лист	№ докум.	Подпись	Дата		

[illegible][illegible]



Приложение А.3 – UML-диаграмма компонентов

№	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----



Приложение Б (обязательное)

Система автотестирования сайтов

Листинг программы

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.09.03.01.08.001 ПЗ

Лист

73

config/sites/saucedemo.config.js

```
module.exports = {
```

```
  name: 'SauceDemo',
```

```
  url: 'https://www.saucedemo.com/',
```

```
  credentials: {
```

```
    standard: { username: 'standard_user', password: 'secret_sauce' },
```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```

        locked:    { username: &apos;locked_out_user&apos;;, password:
&apos;secret_sauce&apos;; }

    },

    selectors: {

        usernameInput:    &apos;[data-test="username"]&apos;;,

        passwordInput:    &apos;[data-test="password"]&apos;;,

        loginButton:      &apos;[data-test="login-button"]&apos;;,

        inventoryContainer: &apos;[data-test="inventory-
container"]&apos;;,

        errorMessage:     &apos;[data-test="error"]&apos;;,

        addToCartBackpack: &apos;[data-test="add-to-cart-sauce-labs-
backpack"]&apos;;,

        cartLink:          &apos;[data-test="shopping-cart-
link"]&apos;;,

        cartItem:          &apos;.cart_item&apos;;,

        removeBackpack:    &apos;[data-test="remove-sauce-labs-
backpack"]&apos;;,

        sortDropdown:      &apos;[data-test="product-sort-
container"]&apos;;,

        itemPrice:         &apos;.inventory_item_price&apos;;

    }

};

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

config/sites/orangehrm.config.js

```
module.exports = {  
  
  name: 'OrangeHRM',  
  
  url: 'https://opensource-demo.orangehrmlive.com/',  
  
  credentials: {  
  
    admin: { username: 'Admin', password:  
'admin123' }  
  
  },  
  
  selectors: {  
  
    usernameInput: '[name="username"]',  
  
    passwordInput: '[name="password"]',  
  
    loginButton: 'button[type="submit"]',  
  
    dashboardContainer: '.oxd-layout',  
  
    errorMessage: '.oxd-alert'  
  
  }  
  
};
```

config/sites/taskmanager.config.js

```
module.exports = {  
  
  name: 'TaskManager',  
  
  url: 'http://localhost:3000',  
  
  credentials: {
```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

```

    valid: { username: &apos;testuser&apos;;, password:
&apos;testpass&apos; }

    },

    selectors: {

        usernameInput:    &apos;[data-test="username-input"]&apos;;,
        passwordInput:    &apos;[data-test="password-input"]&apos;;,
        loginButton:      &apos;[data-test="login-button"]&apos;;,
        errorMessage:     &apos;[data-test="error-message"]&apos;;,
        dashboardContainer: &apos;.dashboard-card&apos;;,
        newTaskInput:      &apos;[data-test="new-task-input"]&apos;;,
        addTaskButton:     &apos;[data-test="add-task-button"]&apos;;,
        taskList:          &apos;[data-test="task-list"]&apos;;,
        logoutButton:      &apos;[data-test="logout-button"]&apos;;

    }

};

```

utils/configLoader.js

```

const saucedemoConfig =
require(&apos;../config/sites/saucedemo.config&apos;);

const orangehrmConfig =
require(&apos;../config/sites/orangehrm.config&apos;);

const taskmanagerConfig =
require(&apos;../config/sites/taskmanager.config&apos;);

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

function getConfig(projectName) {

    switch (projectName) {

        case 'SauceDemo': return saucedemoConfig;

        case 'OrangeHRM': return orangehrmConfig;

        case 'TaskManager': return taskmanagerConfig;

        default: throw new Error(`Неизвестный проект:
${projectName}`);

    }

}

module.exports = { getConfig };

```

pages/LoginPage.js

```

class LoginPage {

    constructor(page, config) {

        this.page = page;

        this.config = config;

    }

    async goto() { await this.page.goto(this.config.url); }

    async login(username, password) {

        await this.page.fill(this.config.selectors.usernameInput,
username);

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

        await this.page.fill(this.config.selectors.passwordInput,
password);

        await this.page.click(this.config.selectors.loginButton);

    }

    async getErrorMessageText() {

        return await
this.page.textContent(this.config.selectors.errorMessage);

    }

    async isErrorMessageVisible() {

        return await
this.page.isVisible(this.config.selectors.errorMessage);

    }

}

module.exports = { LoginPage };

```

pages/InventoryPage.js

```

class InventoryPage {

    constructor(page, config) {

        this.page = page;

        this.config = config;

    }

    async addFirstItemToCart() {

```

```

        await
this.page.click(this.config.selectors.addToCartBackpack);

    }

    async goToCart() {

        await this.page.click(this.config.selectors.cartLink);

    }

    async sortByPriceLowToHigh() {

        await
this.page.selectOption(this.config.selectors.sortDropdown,
        &apos;lohi&apos;);

    }

    async getPricesAsNumbers() {

        const priceElements = await
this.page.$$ (this.config.selectors.itemPrice);

        const pricesText = await Promise.all(priceElements.map(el =>
el.textContent()));

        return pricesText.map(p => parseFloat(p.replace(&apos;$&apos;,
        &apos;&apos;)));

    }

}

module.exports = { InventoryPage };

```

pages/CartPage.js

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

class CartPage {

    constructor(page, config) {

        this.page = page;

        this.config = config;

    }

    async getCartItemCount() {

        return await this.page.$$eval(this.config.selectors.cartItem,
items => items.length);

    }

    async removeFirstItem() {

        await this.page.click(this.config.selectors.removeBackpack);

    }

}

module.exports = { CartPage };

```

pages/TaskManagerPage.js

```

class TaskManagerPage {

    constructor(page, config) {

        this.page = page;

        this.config = config;

    }

    async goto() { await this.page.goto(this.config.url); }

}

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

    async login(username, password) {

        await this.page.fill(this.config.selectors.usernameInput,
username);

        await this.page.fill(this.config.selectors.passwordInput,
password);

        await this.page.click(this.config.selectors.loginButton);

    }

    async getErrorMessage() {

        return await
this.page.textContent(this.config.selectors.errorMessage);

    }

    async addTask(taskText) {

        await this.page.fill(this.config.selectors.newTaskInput,
taskText);

        await this.page.click(this.config.selectors.addTaskButton);

    }

    async getTasks() {

        return await this.page.$$eval(this.config.selectors.taskList +
'<li>;', items =>

            items.map(li => li.firstChild.textContent.trim())

        );

    }

    async deleteTask(index) {

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						82
Изм.	Лист	№ докум.	Подпись	Дата		

```

        await this.page.click(`[data-test="delete-task-${index}"]`);
    }

    async logout() {

        await this.page.click(this.config.selectors.logoutButton);

    }

}

module.exports = { TaskManagerPage };

```

playwright.config.js

```

const { defineConfig } = require('@playwright/test');

require('@dotenv/config');

module.exports = defineConfig({

    testDir: './tests',

    fullyParallel: false,

    retries: 0,

    reporter: 'html',

    use: {

        headless: false,

        viewport: { width: 1280, height: 720 },

        screenshot: 'only-on-failure',

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						83
Изм.	Лист	№ докум.	Подпись	Дата		

```

        video: 'retain-on-failure';

    },

    projects: [

        {

            name: 'SauceDemo',

            testMatch: /(login|cart|sort)\.spec\.js/,

            use: { baseUrl: 'https://www.saucedemo.com/' }

        },

        {

            name: 'OrangeHRM',

            testMatch: /login\.spec\.js/,

            use: { baseUrl: 'https://opensource-
demo.orangehrmlive.com/' }

        },

        {

            name: 'TaskManager',

            testMatch: /taskmanager\.spec\.js/,

            use: { baseUrl: 'http://localhost:3000' }

        }

    ]

});

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

tests/login.spec.js

```
const { test, expect } = require('playwright/test');

const { getConfig } = require('../utils/configLoader');

const { LoginPage } = require('../pages/LoginPage');

test.describe('Авторизация', () => {

  test('Успешная авторизация', async ({ page },
testInfo) => {

    const config = getConfig(testInfo.project.name);

    const loginPage = new LoginPage(page, config);

    await loginPage.goto();

    let validCred = (config.name === 'SauceDemo') ?
config.credentials.standard : config.credentials.admin;

    await loginPage.login(validCred.username, validCred.password);

    const mainContainer = config.selectors.inventoryContainer ||
config.selectors.dashboardContainer;

    await expect(page.locator(mainContainer)).toBeVisible();

  });

  test('Отказ в авторизации (негативный сценарий)',
async ({ page }, testInfo) => {

    const config = getConfig(testInfo.project.name);

    const loginPage = new LoginPage(page, config);
```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

    await loginPage.goto();

    let invalidCred;

    if (config.name === 'SauceDemo') {

        invalidCred = config.credentials.locked;

    } else {

        invalidCred = { username: 'Admin', password:
'wrong_password' };

    }

    await loginPage.login(invalidCred.username,
invalidCred.password);

    if (config.name === 'OrangeHRM') {

        await expect(page).toHaveURL(/\/auth\/\/login/);

    } else {

        await
expect(page.locator(config.selectors.errorMessage)).toBeVisible();

        const errorText = await loginPage.getErrorMessageText();

        expect(errorText).toContain('locked out');

    }

});

});

```

tests/cart.spec.js

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

const { test, expect } = require('playwright/test');

const { getConfig } = require('utils/configLoader');

const { LoginPage } = require('pages/LoginPage');

const { InventoryPage } =
require('pages/InventoryPage');

const { CartPage } = require('pages/CartPage');

test.describe('Корзина (только SauceDemo)', () => {

  test.beforeEach(async ({ page }, testInfo) => {

    const config = getConfig(testInfo.project.name);

    if (config.name !== 'SauceDemo') test.skip(true,
'Только SauceDemo');

  });

  test('Добавление товара в корзину', async ({ page },
testInfo) => {

    const config = getConfig(testInfo.project.name);

    const loginPage = new LoginPage(page, config);

    const inventoryPage = new InventoryPage(page, config);

    const cartPage = new CartPage(page, config);

    await loginPage.goto();

    await loginPage.login(config.credentials.standard.username,
config.credentials.standard.password);

    await inventoryPage.addFirstItemToCart();
  });
});

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```

    await inventoryPage.goToCart();

    expect(await cartPage.getCartItemCount()).toBe(1);

  });

  test('Удаление товара из корзины', async ({ page },
testInfo) => {

    const config = getConfig(testInfo.project.name);

    const loginPage = new LoginPage(page, config);

    const inventoryPage = new InventoryPage(page, config);

    const cartPage = new CartPage(page, config);

    await loginPage.goto();

    await loginPage.login(config.credentials.standard.username,
config.credentials.standard.password);

    await inventoryPage.addFirstItemToCart();

    await inventoryPage.goToCart();

    await cartPage.removeFirstItem();

    expect(await cartPage.getCartItemCount()).toBe(0);

  });

});

```

tests/sort.spec.js

```

const { test, expect } = require('@playwright/test');

const { getConfig } = require('../utils/configLoader');

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

const { LoginPage } = require(&apos;../pages/LoginPage&apos;);

const { InventoryPage } =
require(&apos;../pages/InventoryPage&apos;);

test.describe(&apos;Сортировка товаров (только SauceDemo)&apos;;,
() => {

    test(&apos;Сортировка по цене (low to high)&apos;;, async ({ page
}, testInfo) => {

        const config = getConfig(testInfo.project.name);

        if (config.name !== &apos;SauceDemo&apos; ) test.skip(true,
&apos;Только SauceDemo&apos;);

        const loginPage = new LoginPage(page, config);

        const inventoryPage = new InventoryPage(page, config);

        await loginPage.goto();

        await loginPage.login(config.credentials.standard.username,
config.credentials.standard.password);

        await inventoryPage.sortByPriceLowToHigh();

        const prices = await inventoryPage.getPricesAsNumbers();

        const isSorted = prices.every((val, i, arr) => i === 0 || val
>= arr[i - 1]);

        expect(isSorted).toBe(true);

    });

});

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

tests/taskmanager.spec.js

```
const { test, expect } = require('@playwright/test');

const { getConfig } = require('../utils/configLoader');

const { TaskManagerPage } =
require('../pages/TaskManagerPage');

test.describe('Task Manager', () => {

  let pageManager;

  test.beforeEach(async ({ page }, testInfo) => {

    const config = getConfig(testInfo.project.name);

    pageManager = new TaskManagerPage(page, config);

    await pageManager.goto();

  });

  test('Успешный вход', async () => {

    await pageManager.login('testuser',
'testpass');

    await expect(pageManager.page.locator('.dashboard-
card')).toBeVisible();

  });

  test('Отказ во входе с неверным паролем', async () =>
{

    await pageManager.login('testuser',
'wrongpass');

  });

});
```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

    const errorText = await pageManager.getErrorMessage();

    expect(errorText).toContain('Неверный логин или пароль');
    ;

    });

    test('Добавление новой задачи', async () => {

        await pageManager.login('testuser',
        'testpass');

        await pageManager.addTask('Купить хлеб');

        const tasks = await pageManager.getTasks();

        expect(tasks).toContain('Купить хлеб');

    });

    test('Удаление задачи', async () => {

        await pageManager.login('testuser',
        'testpass');

        await pageManager.addTask('Задача для удаления');

        await pageManager.deleteTask(0);

        const tasks = await pageManager.getTasks();

        expect(tasks).not.toContain('Задача для удаления');

    });

    test('Выход из системы', async () => {

        await pageManager.login('testuser',
        'testpass');

```

```

        await pageManager.logout();

        await
expect(pageManager.page.locator('form#loginForm')).toBeV
isible();

    });

});

```

index.html (страница входа)

```

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Task Manager – вход</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="login-container">
        <div class="card">
            <h2> Task Manager</h2>
            <div class="demo-info">
                <p> Тестовые данные для входа:</p>
                <div class="creds">
                    <span> Логин: <strong>testuser</strong></span>
                    <span> Пароль:
<strong>testpass</strong></span>
                </div>
            </div>
            <form id="loginForm">
                <input type="text" id="username"
placeholder="Логин" data-test="username-input">
                <input type="password" id="password"
placeholder="Пароль" data-test="password-input">

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

        <button type="submit" data-test="login-
button">Войти</button>
        <div id="errorMsg" class="error" data-test="error-
message" style="display:none;"></div>
    </form>
</div>
</div>
<script src="app.js"></script>
</body>
</html>

```

dashboard.html (главная страница)

```

<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Task Manager – задачи</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="dashboard-container">
        <div class="dashboard-card">
            <div class="header">
                <h2> Мои задачи</h2>
                <button id="logoutBtn" data-test="logout-button"
class="logout-btn">Выйти</button>
            </div>
            <div class="add-task">
                <input type="text" id="newTask" placeholder="Что
нужно сделать?" data-test="new-task-input">
                <button id="addTaskBtn" data-test="add-task-
button"> Добавить</button>
            </div>

```

```

        <ul id="taskList" data-test="task-list"></ul>
    </div>
</div>
<script src="app.js"></script>
</body>
</html>

```

Листинг style.css

```

* { margin: 0; padding: 0; box-sizing: border-box; font-family:
'Segoe UI', sans-serif; }
body {
    min-height: 100vh;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    display: flex;
    justify-content: center;
    align-items: center;
}
.card, .dashboard-card {
    background: white;
    border-radius: 20px;
    box-shadow: 0 20px 35px rgba(0,0,0,0.2);
    padding: 30px;
    width: 100%;
    max-width: 500px;
}
.dashboard-card { max-width: 700px; }

```

app.js

```

const VALID_USER = { username: 'testuser', password: 'testpass' };

// Страница логина
if (document.getElementById('loginForm')) {
    const form = document.getElementById('loginForm');

```

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

```

const errorDiv = document.getElementById('errorMsg');
form.addEventListener('submit', (e) => {
    e.preventDefault();
    const username =
document.getElementById('username').value;
    const password =
document.getElementById('password').value;
    if (username === VALID_USER.username && password ===
VALID_USER.password) {
        localStorage.setItem('loggedIn', 'true');
        window.location.href = 'dashboard.html';
    } else {
        errorDiv.textContent = 'Неверный логин или пароль';
        errorDiv.style.display = 'block';
    }
});
}

// Страница дашборда
if (document.getElementById('taskList')) {
    if (localStorage.getItem('loggedIn') !== 'true')
window.location.href = 'index.html';

    let tasks = JSON.parse(localStorage.getItem('tasks')) || [];
    const taskList = document.getElementById('taskList');
    const newTaskInput = document.getElementById('newTask');
    const addBtn = document.getElementById('addTaskBtn');
    const logoutBtn = document.getElementById('logoutBtn');

    function renderTasks() {
        taskList.innerHTML = '';
        tasks.forEach((task, index) => {
            const li = document.createElement('li');
            const taskSpan = document.createElement('span');
            taskSpan.textContent = task;

```

```

        taskSpan.style.cursor = 'pointer';
        taskSpan.style.flex = '1';
        taskSpan.addEventListener('click', () => {
            if (taskSpan.style.textDecoration === 'line-
through') {
                taskSpan.style.textDecoration = 'none';
                taskSpan.style.opacity = '1';
            } else {
                taskSpan.style.textDecoration = 'line-
through';
                taskSpan.style.opacity = '0.6';
            }
        });
        const delBtn = document.createElement('button');
        delBtn.textContent = 'Удалить';
        delBtn.setAttribute('data-test', `delete-task-
${index}`);
        delBtn.addEventListener('click', (e) => {
            e.stopPropagation();
            tasks.splice(index, 1);
            localStorage.setItem('tasks',
JSON.stringify(tasks));
            renderTasks();
        });
        li.appendChild(taskSpan);
        li.appendChild(delBtn);
        taskList.appendChild(li);
    });
}

addBtn.addEventListener('click', () => {
    const newTask = newTaskInput.value.trim();
    if (newTask) {
        tasks.push(newTask);
    }
});

```

```

        localStorage.setItem('tasks', JSON.stringify(tasks));
        newTaskInput.value = '';
        renderTasks();
    }
});

logoutBtn.addEventListener('click', () => {
    localStorage.removeItem('loggedIn');
    window.location.href = 'index.html';
});
renderTasks();
}

```

Приложение В (обязательное)

Система автотестирования сайтов

Презентация

					ПГУ 1.09.03.01.08.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

Система автотестирования сайтов

Автор работы: студент гр.22BBB1 Осмаев М-А.Х.

Научный руководитель: д.т.н. Митрохин М.А.

Что такое автотесты?



Выбор инструмента среди аналогов

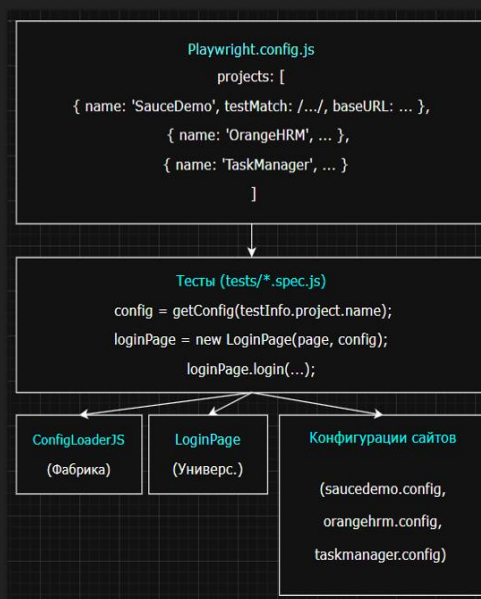


Сравнение инструментов

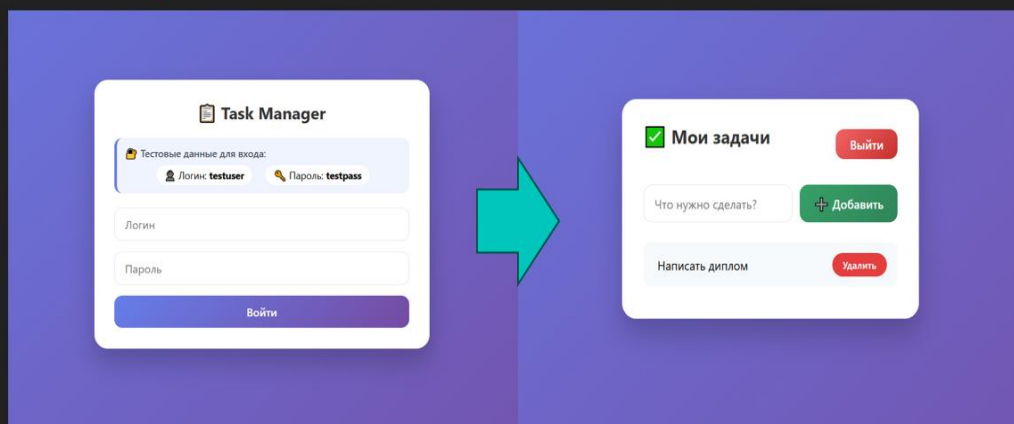
Характеристика	Cypress	Selenium	Playwright
Год выпуска	2014	2004	2020
Поддерживаемые языки	JavaScript, TypeScript	Java, Python, C#, JavaScript, Ruby и др.	JavaScript, TypeScript, Python, Java, .NET
Браузеры	Chrome	Все основные браузеры	Chromium, Firefox, WebKit
Мультивкладки	Нет	Да	Да
Мобильное тестирование	Ограничено	Да (Appium)	Да (эмуляция)
Автоожидания	Да	Нет (требуется явные ожидания)	Да
Параллельное выполнение	Ограничено	Да (Grid)	Да

Архитектура гибкой системы

Для каждого сайта мы делаем отдельный конфигурационный файл



Разработанный мной сайт



Изм.	Лист	№ докум.	Подпись	Дата

Результаты тестов

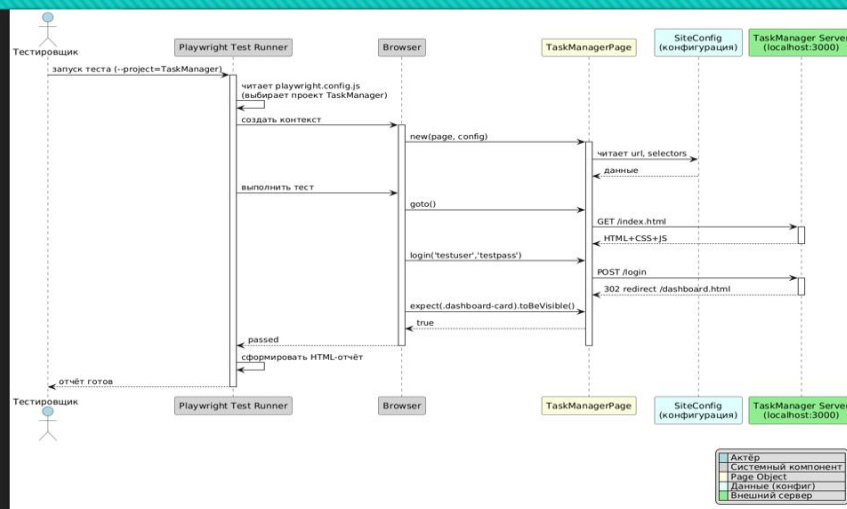
Выводы: для SauceDemo

для OrangeHRM

для TaskManager

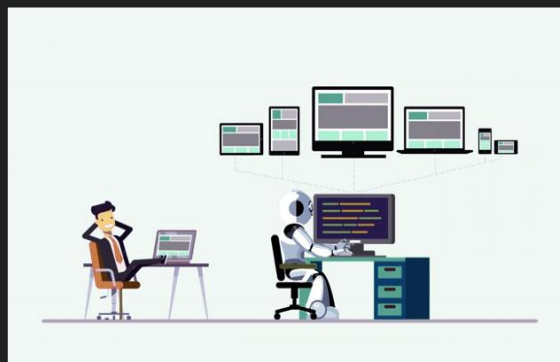
✓ <u>сайт:рес.js</u>		
✓ Корзина (только SauceDemo) - Добавление товара в корзину SauceDemo	сайт:рес.js:13	2.6s
✓ Корзина (только SauceDemo) - Удаление товара из корзины SauceDemo	сайт:рес.js:30	2.1s
✓ <u>login:рес.js</u>		
✓ Авторизация - Успешная авторизация SauceDemo	login:рес.js:6	2.7s
✓ Авторизация - Отказ в авторизации (негативный сценарий) SauceDemo	login:рес.js:23	2.1s
✓ <u>sort:рес.js</u>		
✓ Сортировка товаров (только SauceDemo) - Сортировка по цене (low to high) SauceDemo	sort:рес.js:7	2.9s
✓ <u>login:рес.js</u>		
✓ Авторизация - Успешная авторизация OrangeHRM	login:рес.js:6	6.3s
✓ Авторизация - Отказ в авторизации (негативный сценарий) OrangeHRM	login:рес.js:23	3.9s
✓ <u>taskmanager:рес.js</u>		
✓ Task Manager - собственное приложение - Успешный вход TaskManager	taskmanager:рес.js:13	778ms
✓ Task Manager - собственное приложение - Отказ во входе с неверным паролем TaskManager	taskmanager:рес.js:19	594ms
✓ Task Manager - собственное приложение - Добавление новой задачи TaskManager	taskmanager:рес.js:24	726ms
✓ Task Manager - собственное приложение - Удаление задачи TaskManager	taskmanager:рес.js:31	757ms
✓ Task Manager - собственное приложение - Выход из системы TaskManager	taskmanager:рес.js:39	726ms

Диаграмма последовательности



Где можно применить?

- IT компании
- Учебные курсы по тестированию
- Как основу для дальнейших доработок



СПАСИБО ЗА ВНИМАНИЕ

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.08.001 ПЗ

Лист

102