

Automatisierte Oberflächentests für die S-Erleben-Website der Kreissparkasse Köln

Anforderungen, Implementierung und
Einbindung von Selenium Tests

PRAXISPROJEKT

ausgearbeitet von

Leonie Maleen Eichler

vorgelegt an der

TECHNISCHEN HOCHSCHULE KÖLN
CAMPUS GUMMERSBACH
FAKULTÄT FÜR INFORMATIK UND
INGENIEURWISSENSCHAFTEN

im Studiengang
MEDIENINFORMATIK

Prüfer: Prof. Dr. Frank Victor
Technische Hochschule Köln

Betreuer: Stefan Schulze
OEVERTMANN Networks GmbH

Gummersbach, den 19. Januar 2022

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Gummersbach, den 19. Januar 2022

A handwritten signature in black ink, appearing to read "L. Eichler".

Leonie Eichler

Inhaltsverzeichnis

1 Einleitung	2
1.1 Projekt	2
1.2 S-Erleben-Website der Kreissparkasse Köln	3
1.3 Aktuelles Testverfahren	5
1.4 Projektspezifische Anforderungen	6
1.4.1 TÜV und ISO Zertifizierung	7
2 Testautomatisierungstool für Webanwendungen	9
2.1 Anforderungen	9
2.2 Selenium	9
2.2.1 Selenium IDE	10
2.2.2 Selenium WebDriver	10
2.2.3 Selenium Grid	12
2.2.4 Wahl des Tools	12
3 Automatisierte Tests	13
3.1 Vorbereitung	13
3.1.1 WebDriver	13
3.1.2 Test-Framework und NuGet	13
3.2 Implementierung	14
3.2.1 Prozess zum Ausfüllen des unangemeldeten Kontaktformulars . .	15
3.2.2 Strukturierung eines Tests	17
3.3 MSTest-Frameworks	27
3.3.1 Attribute	27
3.3.2 Assert	29
3.3.3 TestContext	30
3.4 Runsettings	30
3.4.1 Aufbau der Laufzeiteinstellungsdatei	31
3.5 Refactoring	32
3.5.1 Refaktoriertes Arrange	32
3.5.2 Refaktoriertes Act	32
3.5.3 Auslagern von Methoden	35
3.6 Ausführen von Tests	36
3.7 Angepasstes Reporting	36
3.7.1 try-catch für Methoden	37
3.7.2 Screenshot aufnehmen	37
3.8 Weitere Tests	38
3.8.1 Testmethoden Klasse	39
3.8.2 Beispiel Shop-Test	40

Inhaltsverzeichnis

4 Einbindung in das bestehende Projekt	44
4.1 Azure DevOps	44
4.2 Einbindung in einen Release	44
4.3 Überprüfung der Testresultate	45
4.3.1 Detaillierte Testresultate	45
4.4 Auftretende Fehler	46
4.4.1 Session not created	46
4.4.2 No such element	47
4.4.3 Sicherheitszertifikat abgelaufen	48
4.4.4 Testkonten verfügen nicht über ausreichende Berechtigungen	49
4.4.5 Kontaktformulare werden nicht gesendet	49
4.4.6 Client- und Serverfehler	49
4.4.7 Darstellung fehler	50
5 Fazit und Aussicht	51
5.1 Fazit	51
5.2 Aussicht	52
5.2.1 Tests für weitere Institute	52
5.2.2 Tests für die neue Webseite	53
5.2.3 Testumgebung	53
5.2.4 Tests für weitere Websites	53
5.3 Themen für Praxisprojekte und Bachelorarbeiten	53
Abbildungsverzeichnis	54
Tabellenverzeichnis	55
Literaturverzeichnis	60

1 Einleitung

Im folgenden Kapitel wird das Projekt rund um die Sparkasse-Erleben (S-Erleben) Webpräsenz der Kreissparkasse Köln (KSK) von Oevermann Networks GmbH (Oevermann) vorgestellt. Es wird auf das aktuelle Testverfahren für die S-Erleben-Website eingegangen und darauf, welche Anforderungen sich durch das vorhandene Projekt ergeben.

1.1 Projekt

Die Abteilung Intranet/ Extranet Development der Digitalagentur Oevermann Networks GmbH realisiert für das Dienstleistungsunternehmen S-Markt und Mehrwert GmbH & Co KG (SMM) den Webauftritt und die Implementierung des Marketingauftritts für 100 Institute der Sparkasse.

Die Website der Großsparkasse Köln wird seit 2015 mit einer TÜV-Zertifizierung für das Online-Banking entwickelt und ist eine der Top-Referenzen von Oevermann (OEVERMANN Networks GmbH, 2021b).

Den Auftrag hatte die Tochtergesellschaft der Sparkassen-Finanzgruppe S-Institut für Marketing & Kundenbindung (S-IMK) erteilt. 2018 wurde die S-IMK in die zuvor aus der MehrWert Servicegesellschaft mbH (MWSG) und der S Direkt-Marketing GmbH & Co. KG (S Direkt) fusionierten S-Markt & Mehrwert GmbH & Co. KG (SMM) integriert (vgl. DSV Gruppe, 3011; S-Markt & Mehrwert GmbH & Co. KG, 1812). Der Auftrag blieb mit der SMM als neue Auftraggeberin bestehen.

1.2 S-Erleben-Website der Kreissparkasse Köln

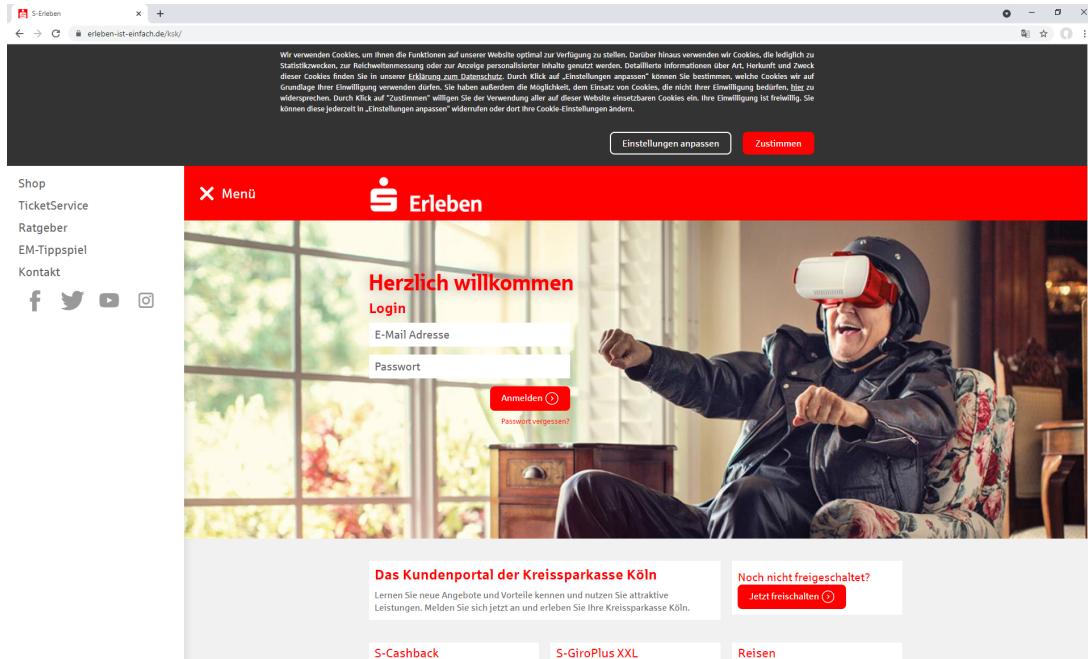


Abbildung 1.1: S-Erleben Webseite

Die SMM bietet Privatkund:innen auf ihrer S-Erleben-Website Produkte, Services und Dienstleistungen im Bereich Banking, Finanzierung und Versicherung. Unter anderem werden Cashback mit 4643 Partnern, eine Reisepreis-Rückvergütung und ein TicketService angeboten. Weitere Dienstleistungen finden sich in der Auflistung der Menüpunkte (s. u.). Für das Onlinebanking registrierte Kund:innen der KSK haben die Möglichkeit S-Erleben für ihr KSK-Konto freizuschalten und sich dann mit einem Anmeldenamen oder ihrer E-Mail Adresse und einer PIN anzumelden (siehe Abbildung 1.1). Vorausgesetzt wird, dass Kunden:innen ab dem 14. Lebensjahr ein Privatgirokonto bei der KSK führen. Dabei sind Basis- und Pfändungsschutzkonten ausgenommen (vgl. S-Markt & Mehrwert GmbH & Co. KG, 2019, S. 1).

Im angemeldeten Bereich werden die Menüpunkte aus der folgenden Liste mit den ggf. vorhandenen Unterpunkten aufgelistet, die die Nutzer:innen auf die entsprechenden Webseiten weiterleiten (siehe Abbildung 1.2).

1 Einleitung

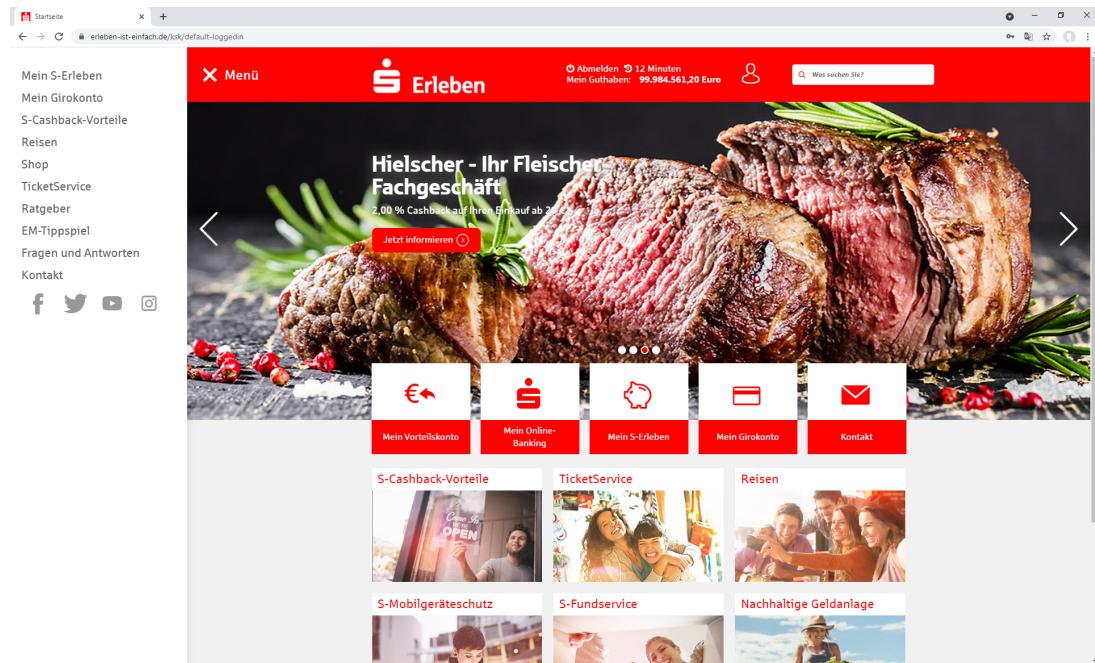


Abbildung 1.2: S-Erleben Webseite eingeloggt

Menüpunkte der S-Erleben-Website:

- Mein S-Erleben
 - Einstellungen
 - Vorteilskonto
- Mein Girokonto
 - Girokonto-Leistungen
 - Online-Banking
 - Kreditkarten-Leistungen
 - S-Mobilgeräteschutz
 - S-Fundservice
 - ISIC-Studentenausweis
- S-Cashback-Vorteile
 - S-Cashback Regional
 - S-Cashback Online
 - Cashback auszahlen
 - Cashback einlösen
- Reisen

- Shop
 - Meine Bestellungen
- TicketService
- Ratgeber
- EM-Tippspiel
- Fragen und Antworten
- Kontakt

Die Bereiche unterscheiden sich im unangemeldeten und angemeldeten Zustand. Es kann beispielsweise unangemeldet auf den Shop zugegriffen werden, jedoch gibt es nicht die Möglichkeit, die Artikel in den Warenkorb zu legen. Auf ähnliche Weise unterscheiden sich weitere Bereiche, wie die Kontaktseite. Dies wird ausführlicher in den entsprechenden Tests erläutert (siehe Kap.3).

Die Website wird fortlaufend weiterentwickelt und auftretende Fehler werden behoben. Gleichzeitig befindet sich eine neue Version der Website in der Entwicklung. Für diese sollen vor dem Release die Tests der aktuellen Webseite angepasst und weiterentwickelt werden, falls sie den Anforderungen entsprechen.

1.3 Aktuelles Testverfahren

Zurzeit werden im Zusammenhang mit der S-Erleben-Website drei ausgewählte Funktionen mit Unit-Tests geprüft. Units sind atomare Programmereinheiten, die groß genug sind, um eigenständig getestet werden zu können. In der Praxis können Units einzelne Funktionen, Klassen, Pakete und Bibliotheken umfassen (vgl. Hoffmann, 2013, S. 159).

Es gibt einen Unit-Test für den Partner-Import, bei dem alle lokalen, bundesweiten und online Partner von der Schnittstelle **S-Partnermanager** importiert werden und dies geprüft wird. Einen weiteren Unit-Test gibt es für die Indexerstellung, bei der neue oder geänderte Shop Artikel, Partner, Veranstaltungen, Seiten, Spenden und Angebote geholt werden und in den Index für die Suche geschrieben werden. Der dritte Test ist für die „Einmalanmeldung“ (Single Sign-on) des Onlinebankings. Diese Unit-Tests werden zum jetzigen Zeitpunkt nur verwendet, wenn Partner oder Veranstaltungen in die lokale Datenbank (DB) importiert werden.

Zusätzlich werden, wenn von den Entwickler:innen Änderungen im Zusammenhang mit der S-Erleben-Website der KSK vorgenommen werden, die Bereiche, auf die sich die Änderungen auswirken können, manuell überprüft. Dieses sogenannte *Smoke testing* ist eine zeitaufwändige und fehleranfällige Arbeit. Microsoft (2007) definierte dies in der Dokumentation *Guidelines for Smoke Testing* folgendermaßen: „In software, the term smoke testing describes the process of validating code changes before the changes are checked into the product’s source tree. After code reviews, smoke testing is the most cost effective method for identifying and fixing defects in software. Smoke tests

are designed to confirm that changes in the code function as expected and do not destabilize an entire build.“

Um die Softwarequalität zu verbessern, werden automatisierte Tests für Webanwendungen entwickelt.

1.4 Projektspezifische Anforderungen

Die Tests müssen in das bestehende Projekt der Abteilung eingebunden werden. Da Oevermann als ein Microsoft Gold Certified Partner im Bereich DevOps sowie drei weiteren Kategorien mit einem Silber-Zertifikat ausgezeichnet wurde (Microsoft, 2021) und der Status beibehalten werden soll, muss die Programmiersprache C# verwendet werden.

„C# ist eine allgemein anwendbare, typsichere, objektorientierte Programmiersprache, die die Produktivität des Programmierers erhöhen soll. Zu diesem Zweck versucht die Sprache, die Balance zwischen Einfachheit, Ausdrucksfähigkeit und Performance zu finden. Die Sprache C# ist plattformneutral, wurde aber geschrieben, um gut mit dem .NET Framework von Microsoft zusammenzuarbeiten. C# 7.0 ist auf das .NET Framework 4.6/4.7 ausgerichtet.“ (Albahari u. Albahari, 2018). Im Projekt wird die C# Version 7.3 verwendet. Diese muss auch für die Tests verwendet werden. Das aktuelle .NET Framework mit der Version 4.8 (.NET Blog, 2019) wird bei Oevermann genutzt.

Als letzte Anforderung müssen die fertigen Tests in die Azure DevOps Server des Projekts eingebunden werden.

1.4.1 TÜV und ISO Zertifizierung



Abbildung 1.3: TÜV ISO 27001 Zertifizierungsprozess (TÜV-Rheinland)

Die Banksoftware von OEVERMANN Networks GmbH (2021a) ist vom TÜV Rheinland ISO 27001 zertifiziert. Die DIN EN ISO/IEC 27001 (2017) definiert Anforderungen an die Umsetzung und die Dokumentation eines Informationssicherheitsmanagementsystems (ISMS), worum es sich bei der S-Erleben-Website handelt. Die internen Programmierrichtlinien orientieren sich unter anderem an der ISO 27001 und definieren einen Prozess zum Testen einer Software. Dieser soll durch automatisierte Oberflächentests begleitet werden und somit die testspezifischen Qualitätsanforderungen der DIN EN ISO/IEC 27001 (2017) erfüllen. Zu diesen gehören unter anderem:

- Bereitstellung der erforderlichen Ressourcen für das Testverfahren
- Trennung von Entwicklungs-, Test- und Betriebsumgebung
- Tests sind unabhängig von der Entwicklung

1 Einleitung

- Spezifikation der verwendeten Tools, Entwicklungs- und Testumgebung und der Testfälle
- Trennung von Entwicklungsaufgaben und Tests
- Testdaten müssen besonders geschützt werden

Das verwendete Tool wird in dem Kapitel 2 spezifiziert, die Entwicklungs- und Testumgebung im Abschnitt 1.4 und dem Kapitel 3. In der Phase der Testplanung wurde festgelegt, dass für die in Abschnitt 3.2 aufgelisteten Bereiche Testfälle erstellt werden sollen. Da Testdaten besonders geschützt werden müssen, werden keine sicherheitskritischen Daten, wie Passwörter im Praxisprojekt erwähnt.

Damit das Zertifikat nach ISO 27001:2013 vom TÜV erteilt werden kann, müssen im ersten Schritt eine Bestandsaufnahme und optional ein Voraudit stattfinden (siehe Abbildung 1.2). Daraufhin folgen die Audits der Stufe 1 und 2, bei denen die Dokumentation der Unterlagen, die praktische Anwendung und die Wirksamkeit des Managementsystems bewertet und geprüft werden. Nach der Zertifizierung folgen jährliche Überwachungsaudits, bei denen die Prozessoptimierung und die Normkonformität überprüft wird. Drei Jahre nach dem Erhalt des Zertifikats erfolgt eine Re-Zertifizierung. Diese fand bei Oevermann im Mai 2021 statt.

Aus diesem Grund müssen die Tests sowohl die Anforderungen der ISO 27001:2013, als auch die internen Programmierrichtlinien, die sich an der ISO 27001 orientieren, erfüllen.

2 Testautomatisierungstool für Webanwendungen

Dieses Kapitel definiert die Anforderungen an ein Testautomatisierungstool für Webanwendungen, welches genutzt werden soll. Es wird das Tool *Selenium* vorgestellt.

2.1 Anforderungen

Neben den projektspezifischen Anforderungen (siehe Abschnitt 1.4) gibt es noch zusätzliche Anforderungen an das Testautomatisierungstool. Dieses muss Tests mit den meist genutzten Browsern Chrome, Firefox, Microsoft Edge und Safari durchführen können und gleichzeitig auf den Betriebssystemen Microsoft Windows, Linux und macOS laufen.

Zusätzlich soll es eine Möglichkeit zum Reporting geben. Dabei müssen Entwickler:innen über auftretende Fehler und Abweichungen informiert werden. Da die Software der S-Erleben-Website stetig geändert wird, müssen die Tests leicht zu warten und anzupassen sein. Die Lizenzkosten für die Nutzung des Tools sollen gering gehalten werden, da es von Oevermann kommerziell genutzt wird.

2.2 Selenium

Selenium ist eines von vielen Testautomatisierungstools für Webanwendungen. Das Framework ist open source und wurde 2004 von Jason Huggins bei ThoughtWorks entwickelt (Selenium.16.07.2021b).

„Selenium supports all major browsers and tests can be written in many programming languages and run on Windows, Linux, and Macintosh platforms.“ Zhan (2015). Mittels der Apache Lizenz 2.0 kann Selenium kostenlos für private und kommerzielle Zwecke genutzt werden. Die drei aktuellen Bestandteile von Selenium sind die Selenium IDE (integrierten Entwicklungsumgebung), der Selenium WebDriver und Selenium Grid (vgl. Selenium, 2021d), auf die im Folgenden eingegangen wird.

Plug-ins

Selenium kann zusätzlich durch Plug-ins erweitert werden. Diese werden jedoch nicht von Selenium bereitgestellt, sondern von Drittanbietern/-personen, von denen sie auch verwaltet verwaltet. Selenium verweist darauf, dass diese Plug-ins möglicherweise nicht unter der Apache 2.0 Lizenz laufen und somit nicht kostenlos für private und kommerzielle Zwecke genutzt werden können (vgl. Selenium, 0401).

2.2.1 Selenium IDE

Für diesen Abschnitt wird die (vgl. Selenium IDE, 1205) als Quelle genutzt.

Mit der Selenium IDE können browserbasierte Tests aufgenommen und abgespielt werden. Sie ist eine Erweiterung für die Browser Mozilla Firefox und Google Chrome. Die Aufnahmen der Tests entstehen durch die Interaktion mit der Webanwendung, bei der die IDE diese Aufnahmen in ein Skript wandelt. Diese Skripte können bearbeitet und exportiert werden. Aktuell wird der Export in folgende Programmiersprachen und Testframeworks unterstützt:

- C# NUnit
- Java JUnit
- JavaScript Mocha
- Python pytest

Mit Bedingungen und Schleifen können Ergebnisse geprüft und Abschnitte oder ganze Tests wiederholt werden. Mehrere Tests können zu sogenannten „Suites“ gruppiert werden. Die Tests können unter anderem lokal mit Firefox, Chrome, Safari, Internet Explorer und Edge abgespielt werden. Die Voraussetzung dafür ist, dass die Browser und entsprechenden Browersetreiber installiert sind. Zusätzlich muss für Edge und Internet Explorer Windows genutzt werden.

2.2.2 Selenium WebDriver

Simon Stewart entwickelte ab 2007 den WebDriver (Zhan, 2015). Dieser wurde mit Selenium 2.0, das im Juni 2008 veröffentlicht wurde, zusammengeführt und Selenium WebDriver (beziehungsweise Selenium 2.0) genannt. Im Februar 2021 wurde die Beta 1 für Selenium 4 veröffentlicht, das offiziell am 13. Oktober 2021 veröffentlicht wurde (Selenium, 2021a). Die aktuelle Version ist 4.1.0.

Der WebDriver ist eine objektorientierte Programmierschnittstelle und wird auf der offiziellen Website von Selenium (2021b) folgendermaßen beschrieben:

„Selenium supports automation of all the major browsers in the market through the use of WebDriver. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behaviour of web browsers. Each browser is backed by a specific WebDriver implementation, called a driver. The driver is the component responsible for delegating down to the browser, and handles communication to and from Selenium and the browser.“

Kommunikation

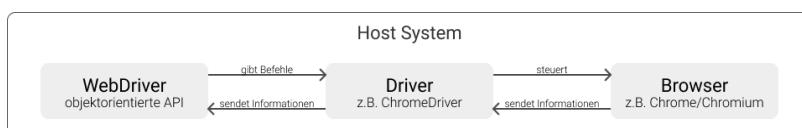


Abbildung 2.1: Selenium: direkte Verbindung (vgl. Selenium, 0712b)

2 Testautomatisierungstool für Webanwendungen

Diese Kommunikation kann entweder lokal oder (remote) auf dem Selenium Server stattfinden. Dadurch kann sie zwischen den einzelnen Komponenten unterschiedlich ablaufen. Der WebDriver ist eine Programmierschnittstelle, die es erlaubt, den Browser mittels Befehlen über den entsprechenden Driver zu steuern. Da die Kommunikation bidirektional ist, werden die Informationen vom Browser an den WebDriver über den Driver zurückgespielt. Dies beschreibt eine direkte Verbindung (siehe Abbildung 2.1).

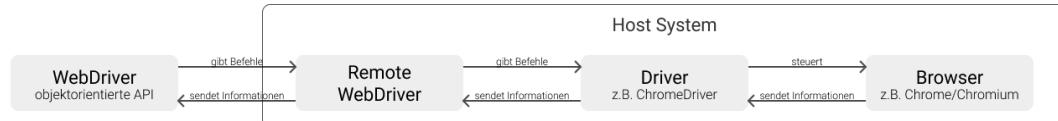


Abbildung 2.2: Selenium: remote Verbindung (vgl. Selenium, 0712b)

Im Gegensatz dazu kommuniziert bei einer remote Verbindung der WebDriver mit einem Selenium Server oder Remote WebDriver, die sich zwischen WebDriver und Driver befinden (siehe Abbildung 2.1). Wenn ein Remote WebDriver verwendet wird, ist der WebDriver als einziges Element nicht im Host-System. Wird der Selenium Server oder Selenium Grid (2.2.3) verwendet, befinden sich diese auch nicht auf dem Host-System.

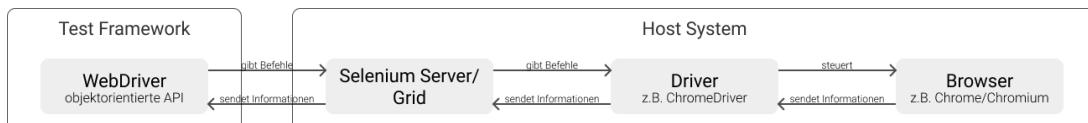


Abbildung 2.3: Selenium: remote Verbindung mit einem Testframework (vgl. Selenium, 0712b)

Da der Browser nur Informationen an den WebDriver zurückspielt, dieser jedoch keine Testfunktionen hat, wird ein Testframework (siehe Kapitel 3.1.2) benötigt. Dieses führt sowohl den WebDriver als auch die Schritte eines Tests aus (siehe Abbildung 2.1).

Offiziell werden vom Selenium Framework folgende Browser mit entsprechenden Drivers unterstützt:

Tabelle 2.1: Von Selenium unterstützte Browser

Browser	Betriebssystem	Verwaltet von
Chromium/Chrome	Windows/macOS/	Google
Firefox	Windows/macOS/Linux	Mozilla
Edge	Windows/macOS	Microsoft
Internet Explorer	Windows	Selenium Project
Safari	macOS High Sierra and newer	Apple
Opera	Windows/macOS/Linux	Opera

Auf die Methoden und Funktionen wird genauer im Kapitel 3 eingegangen und an Beispielen verdeutlicht.

2.2.3 Selenium Grid

Selenium Grid ist die dritte Komponente von Selenium. Mit ihr kann das Ziel verfolgt werden, mehrere WebDriver Skripte parallel auf unterschiedlichen virtuellen oder realen remote Maschinen auszuführen. Dabei können verschiedene Browsersversionen und -konfigurationen zentral verwaltet werden und müssen nicht in jedem Test einzeln definiert sein. Mit Selenium Grid besteht die Möglichkeit Delegations- und Distributionsprobleme zu verringern (vgl. Selenium, 2021c).

2.2.4 Wahl des Tools

Da Selenium somit sowohl die Anforderungen für das Testautomatisierungstool als auch die projektspezifische Anforderungen erfüllt, werden die Tests mit dem Selenium WebDriver erstellt. Für das Praxisprojekt werden die Tests zunächst nur mit dem Chrome Browser erstellt und in C# geschrieben. Dazu mehr im nächsten Kapitel 3.

3 Automatisierte Tests

In diesem Kapitel wird auf die Vorbereitung und Implementierung von den Selenium-tests für die S-Erleben-Website der KSK eingegangen. Dazu wird das Ausfüllen und Absenden eines Kontaktformulars als Beispiel genutzt.

3.1 Vorbereitung

Im Folgenden wird auf die Einrichtung des Selenium WebDrivers für Visual Studio 2019 (VS) und dem Chrome-Browser eingegangen.

3.1.1 WebDriver

Der WebDriver ist ein objektorientierte open source Programmierschnittstelle (API). Diese stellt Funktionen für automatisierte Tests von Webanwendungen. Dazu zählen unter anderem die Navigation im Browser und Benutzereingaben (vgl. Chromium, 3012).

Der ChromeDriver ist eine separat ausführbare Datei, mit der der WebDriver Chrome steuert (vgl. Chromium, 1904). Wie in Tabelle 2.1 zu sehen, kann der ChomeDriver mit den Betriebssystemen Windows, macOS und Linux verwendet werden. Dazu muss Chrome installiert sein und der ChromeDriver muss dieselbe Version wie Chrome haben. Für Selenium 4 muss die Version 75 oder höher verwendet werden (vgl. Selenium, 1212).

3.1.2 Test-Framework und NuGet

In den folgenden Unterabschnitten wird das verwendete Test-Framework und die verwendeten NuGet-Pakete beschrieben.

Test-Framework

Frameworks für (Unit-)Tests führen Tests von einzelnen Komponenten einer Software durch. „Unit-Testing-Frameworks helfen den Entwicklern, die Tests schneller mit einem Satz von bekannten APIs zu schreiben, sie automatisiert auszuführen und die Testresultate leichter zu begutachten.“(Osherove u. a., 2015, S. 43)

Somit unterstützen sie, Tests zu strukturieren, zu warten und auszuwerten. Auch für Tests, die mit Selenium erstellt wurden, können sie verwendet werden.

Für das Praxisprojekt wird das vorgegebene Microsoft Test Framework (MSTest), welches für Komponententests in Visual Studio genutzt werden kann, verwendet. Es ist Teil von Visual Studio. Genauer wird auf das MSTest-Framework im Abschnitt 3.3 eingegangen.

Für Komponenten-Tests können auch andere Frameworks für .NET genutzt werden, wie beispielsweise NUnit.

NuGet

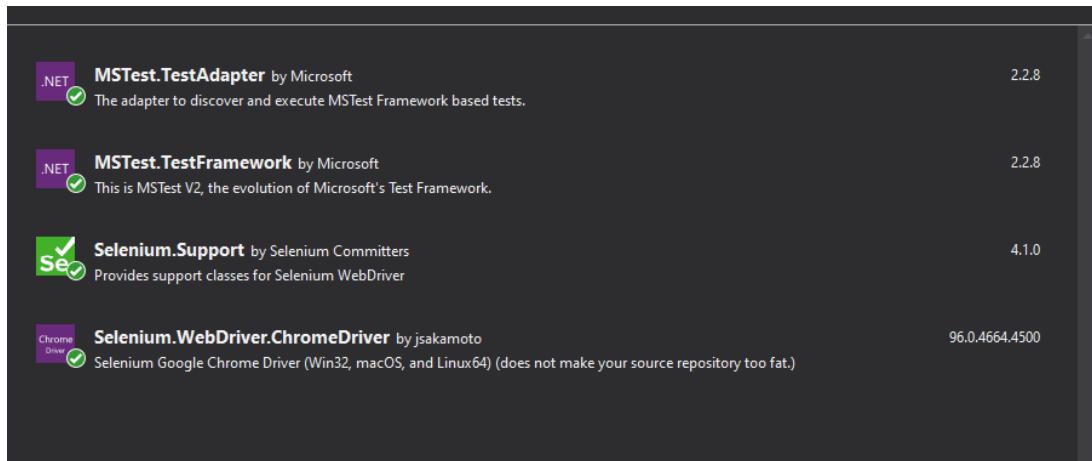


Abbildung 3.1: Verwendete NuGet Pakete

NuGet ist ein Tool von Microsoft, mit dem Software-Komponenten in der Form von Paketen privat oder öffentlich freigegeben und genutzt werden können (vgl. Microsoft, 0212). Für das Praxisprojekt werden die in der Abbildung 3.1 zu sehenden und öffentlich verfügbaren Pakete für MSTest und Selenium verwendet.

3.2 Implementierung

Im Rahmen des Praxisprojekts werden für die folgenden vorgegebenen Bereiche der KSK S-Erleben-Website Tests entwickelt:

- Kontaktformular (unangemeldet)
- Kontaktformular für Cashback Online (angemeldet)
- Onlineshop
- Passwort ändern (angemeldet)
- Kontaktdaten ändern (angemeldet)

Für jeden Test wurde eine eigene Testklasse mit einer Testmethode erstellt. Der Name der Klasse stellt sich aus dem Wort *Test*, dem zu testenden Bereich der Website und dem Kürzel der Sparkasse zusammen. Für die Testmethode wird auf das Kürzel verzichtet.

Die Klasse für das Ausfüllen des unangemeldeten Kontaktformulars 3.5 wird in diesem Kapitel im Detail betrachtet. Bei den anderen Tests wird auf bestimmte Funktionen und Methoden eingegangen.

3.2.1 Prozess zum Ausfüllen des unangemeldeten Kontaktformulars

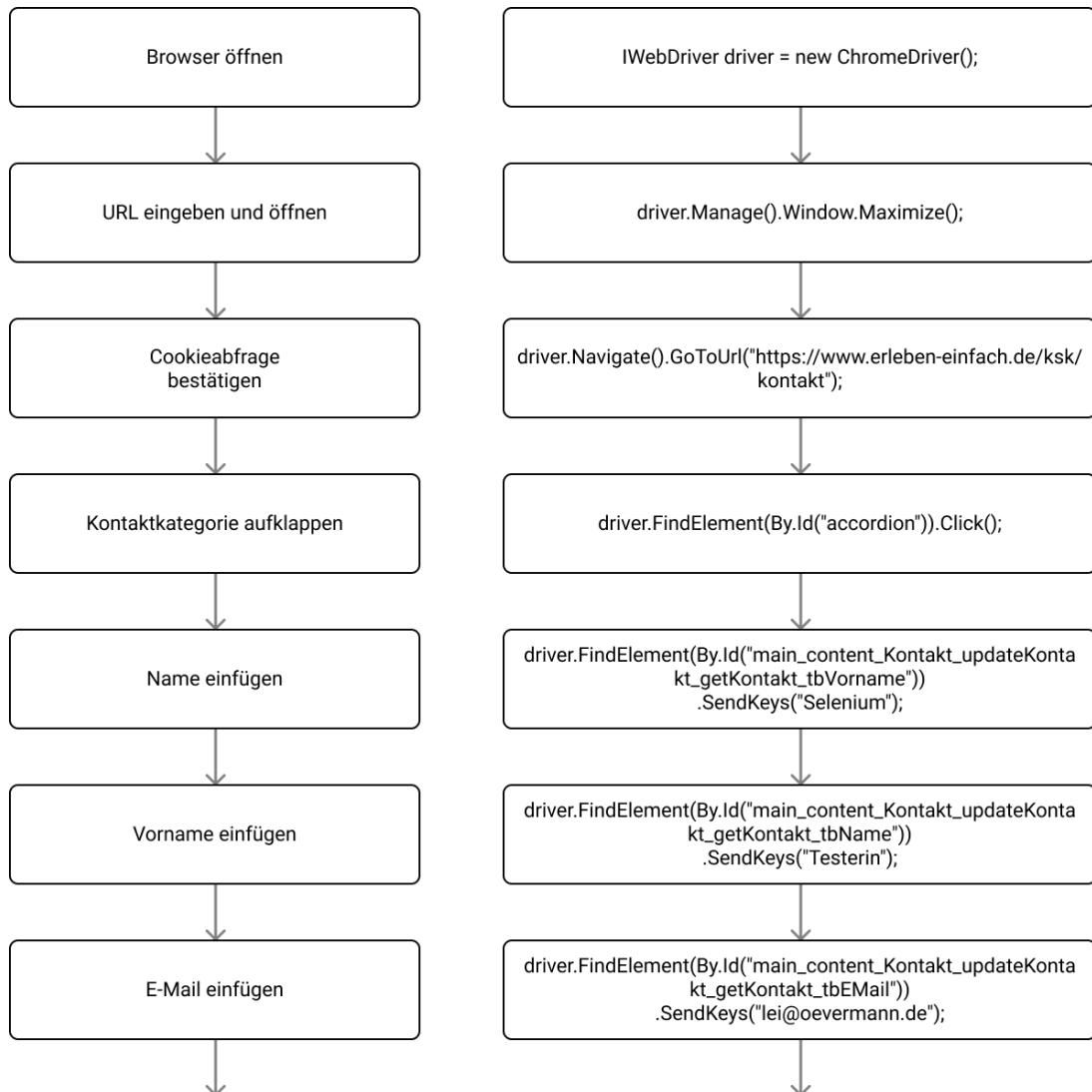


Abbildung 3.2: Prozessdiagramm des Kontaktformulars der S-Erleben-Website (1)

3 Automatisierte Tests

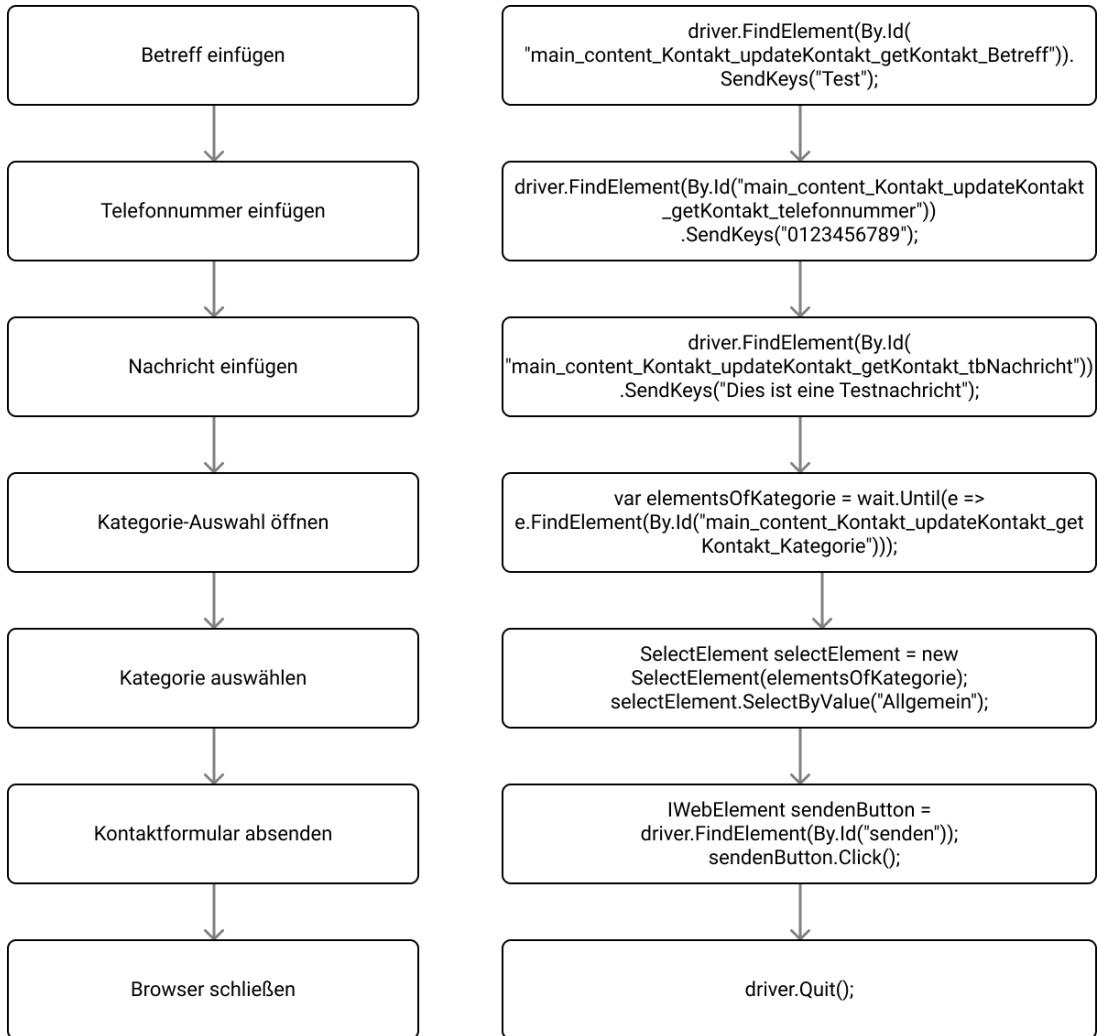


Abbildung 3.3: Prozessdiagramm des Kontaktformulars der S-Erleben-Website (2)

In diesem Unterkapitel wird der Prozess zum Ausfüllen des unangemeldeten Kontaktformulars (siehe Abbildung 3.6) beschrieben. Die Abbildungen 3.2 und 3.3 dienen zur Visualisierung.

Möchte ein:e Nutzer:in online über die S-Erleben-Website mit Verantwortlichen in Kontakt treten, muss zuerst ein Browser geöffnet und daraufhin die URL (Uniform Resource Locator) der Webseite für den Kontakt eingegeben und aufgerufen werden. Unangemeldet kann auf der Seite nur eine Kontaktkategorie ausgewählt werden, die für allgemeine Fragen (siehe Abbildung 3.6). Angemeldet gibt es zusätzlich noch Kontaktformulare für weitere Kategorien, wie die Kategorien Cashback Online.

Nachdem das Formular durch das Anklicken aufgeklappt wurde, können persönliche Daten und die Nachricht in Pflichtfelder eingetragen werden. Durch das Klicken des Buttons **Absenden** wird die Nachricht und die dazugehörigen persönlichen Daten übermittelt. Da eine E-Mail-Adresse mit angegeben werden muss, wird bei einer erfolgreichen Übermittlung der Nachricht eine Bestätigungs-E-Mail gesendet. Für den Test

mit Selenium wird der Vorgang des Ausfüllens und Absendens des Kontaktformulars nachgebaut.

3.2.2 Strukturierung eines Tests

Für die Strukturierung von Test-Klassen wird häufig das AAA-Pattern (dt. Muster) verwendet. AAA steht für Arrange, Act und Assert, also Arrangieren, Agieren und Bestätigen (vgl. Osherove u. a., 2015, S. 53).

Beim Arrangieren werden alle Vorbereitungen getroffen, die für einen Test nötig sind. So werden beispielsweise die Variablen und Objekte erzeugt und initialisiert. Daraufhin werden mit den arrangierten Objekten die Tests ausgeführt und zuletzt überprüft, ob der Test zum entsprechenden Ergebnis führt.

In den nächsten Abschnitten werden erst die Variablen, dann der Testkontext und am Ende die Überprüfung des Ergebnisses implementiert.

Arrange



Abbildung 3.4: Neue Instanz eines ChromeDrivers

Um eine neue Instanz des Chrome-Browsers zu initialisieren, muss eine Instanz des IWebDriver in der Testmethode erstellt werden. Es wird explizit keine Instanz des ChromDrivers erstellt, damit ein zukünftiger Wechsel zwischen den einzelnen Browsern einfacher vorgenommen werden kann und so unterschiedliche Browser getestet werden können.

Für jeden Test sollte ein neuer WebDriver aufgerufen werden, um Seiteneffekte zu vermeiden und unabhängige Tests zu entwickeln (vgl. Selenium, 0712a). Dieser Instanz

wird dann ein neues Objekt der Klasse ChromeDriver zugewiesen. Hierbei stellt die Schnittstelle `IWebDriver` Methoden für den WebDriver zur Verfügung (siehe Tabelle 3.1).

Tabelle 3.1: Methoden der `IWebDriver` Schnittstelle für den WebDriver (vgl. Sel)

Methode	Beschreibung
<code>Close</code>	Schließt das aktuelle Fenster und beendet den Browser, wenn es das letzte aktuell geöffnete Fenster ist.
<code>Dispose</code>	Führt anwendungsdefinierte Aufgaben im Zusammenhang mit dem Freigeben oder Zurücksetzen von nicht verwalteten Ressourcen durch.
<code>.FindElement</code>	Findet das erste <code>IWebElement</code> unter Verwendung der angegebenen Methode.
<code>FindElements</code>	Findet alle <code>IWebElement</code> innerhalb des aktuellen Kontexts unter Verwendung des angegebenen Mechanismus.
<code>Manage</code>	Weist den Driver an, seine Einstellungen zu ändern.
<code>Navigate</code>	Weist den Driver an, den Browser zu einem anderen Ort zu navigieren.
<code>Quit</code>	Beendet diesen Driver und schließt alle zugehörigen Fenster.
<code>SwitchTo</code>	Weist den Treiber an, künftige Befehle an einen anderen Frame oder ein anderes Fenster zu senden.

Erweiterungsmethoden

<code>ExecuteJavaScript</code> <code>(String, Object[])</code>	Führt JavaScript im Kontext des aktuell ausgewählten Frames oder Fensters aus.
<code>ExecuteJavaScript<</code> <code>T >(String, Object[])</code>	Führt JavaScript im Kontext des aktuell ausgewählten Frames oder Fensters aus.
<code>TakeScreenshot</code>	Liefert ein Screenshot-Objekt, das das Bild der Seite auf dem Bildschirm repräsentiert.

Damit die Schnittstelle `IWebDriver` verwendet werden kann, muss in der Datei das Paket `OpenQA.Selenium` eingebunden werden. `OpenQA.Selenium` enthält unter anderem eine Schnittstelle zur Verwaltung von Browsereinstellungen wie `WebDriver.Options` und zur Aufnahme von Bildschirmfotos mittels der `TakeScreenshot`-Schnittstelle.

Zusätzlich enthält es unter anderem Klassen, wie die `By` Klasse, welche genutzt wird, um Elemente in einem Dokument zu lokalisieren, Enums und Exceptions wie die `NoSuchElementException`, die geworfen wird, wenn ein Element nicht gefunden werden kann (vgl. Selenium).

3 Automatisierte Tests

Das Paket `OpenQA.Selenium.Chrome` enthält unter anderem die Klasse `ChromeDriver`, die Chrome steuert oder `ChromeOptions`, die für den `ChromeDriver` spezifische Optionen verwaltet und muss deshalb auch eingebunden werden (vgl. Selenium, 2412b).

```
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;

namespace SIMK.Frontend.SeleniumTest
{
    public class TestKontaktformularKSK
    {
        public void TestKontaktformular()
        {
            IWebDriver driver = new ChromeDriver();
        }
    }
}
```

Die geöffnete Instanz des `ChromeDrivers` ist in Abbildung 3.4 zu sehen.

Act

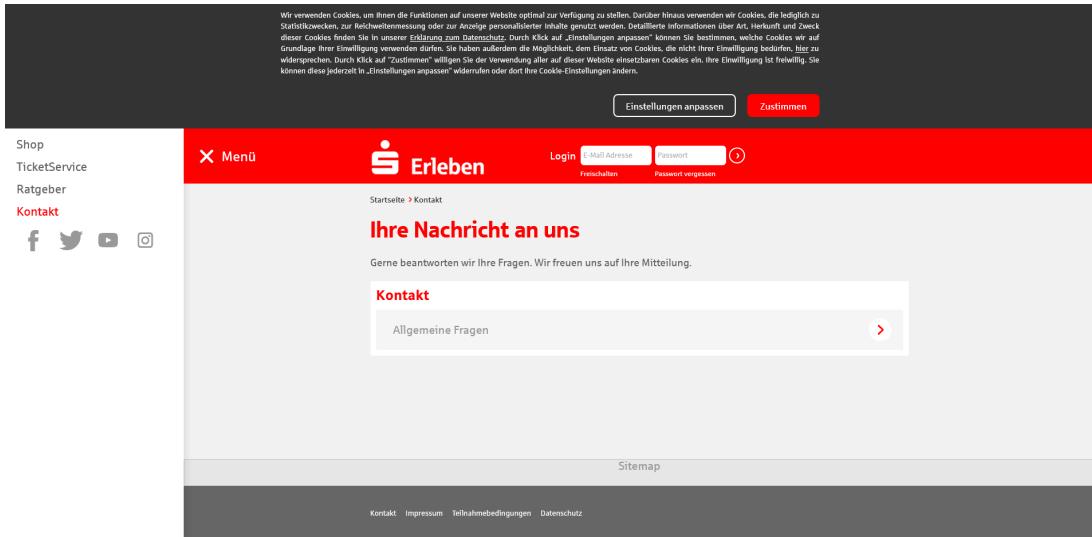


Abbildung 3.5: Kontaktseite der S-Erleben-Website

Über die Methode `Manage()` kann unter anderem die Größe und Position des Browserfensters individuell festgelegt werden. Mit dem Aufruf:

```
driver.Manage().Window.Maximize();
```

3 Automatisierte Tests

wird das Fenster maximiert und füllt für die meisten Betriebssysteme den Bildschirm aus, ohne die Taskleiste zu überlappen.

Um zu einer spezifischen Webseite zu gelangen, muss der Methode `GoToUrl(string url)` dieser URL als Zeichenkette übergeben werden. Der Aufruf, um zur Kontaktseite der KSK S-Erleben-Website zu gelangen, lautet:

```
driver.Navigate().GoToUrl(  
    "https://www.erleben-ist-einfach.de/ksk/kontakt");
```

Nachdem sich der Browser auf der Kontaktseite der KSK S-Erleben-Website befindet (siehe Abbildung 3.6), können die Elemente der Webseite und deren Kindelemente identifiziert und mit ihnen interagiert werden. So können beispielsweise Buttons angeklickt und Inputfelder ausgefüllt oder deren Inhalt gelöscht werden.

Dies ist möglich, da das Dokument, welches beim Aufrufen einer Webseite, in diesem Fall der Kontaktseite der KSK S-Erleben, geladen wird, mit der Auszeichnungssprache (markup language (ML)) Hypertext Markup Language (HTML) strukturiert wird und deshalb die Elemente der Webseite als Objekte in einer baumartigen Struktur, dem Document Object Model (DOM), dargestellt werden (vgl. Jonathan Robie, 2212).

Kontakt

The screenshot shows a contact form titled "Kontaktformular". It contains the following fields:

- Name* (input field)
- Vorname* (input field)
- E-Mail* (input field)
- Betreff* (input field)
- Telefonnummer für Rückfragen* (input field)
- Kategorie* (dropdown menu with placeholder "Wählen Sie eine Kategorie")
- Nachricht* (text area)

At the bottom right is a red button labeled "Absenden" with a circular arrow icon.

* Pflichtfeld

Abbildung 3.6: Kontaktformular der S-Erleben-Website

Durch das Anklicken des Akkordeons *Allgemeine Fragen* (Abbildung 3.5) öffnet sich der Inhalt und das Kontaktformular wird sichtbar (Abbildung 3.6). Für diesen Schritt muss das Akkordeon als Element der Webseite, also Knoten der DOM, lokalisiert werden und daraufhin angeklickt werden.

```
IWebElement element = driver.FindElement(positionsgeber);
```

Mit folgenden Positionsgebern kann ein Webelement durch die Methode `.FindElement()` identifiziert werden:

- class name
- css selector
- id
- name
- link text
- partial link text
- tag name
- xpath

Dazu wird die Klasse `By` genutzt, die `java.lang.Object` erweitert und die Unterklassen:

- `By.ByClassName`
- `By.ByCssSelector`
- `By.ById`
- `By.ByLinkText`
- `By.ByName`
- `By.ByPartialLinkText`
- `By.ByTagName`
- `By.ByXPath,`
- `ByAll`
- `ByChained`
- `ByIdOrName`
- `RelativeLocator.RelativeBy`

hat (Selenium, 2412a).

Der Bereich (engl. division (DIV)) des Akkordeons wird definiert durch:

```
<div id="accordion" class="ui-accordion ui-widget ui-helper-reset"  
role="tablist"></div>
```

Als Positionsgeber könnte in diesem Fall die `id`, die `class` und die `role` genutzt werden. Der Funktionsaufruf mit der Id lautet:

```
IWebElement accordionElement = driver.FindElement(By.Id("accordion"));
```

Hierbei wird ein neues Objekt des Typs `IWebElement` initialisiert, dass die Instanz des WebDrivers nutzt, um die Methode `.FindElement()` aufzurufen, dem die Id als Zeichenkette übergeben wird. Der Aufruf für die Funktion mit dem Klassennamen ist:

```
IWebElement accordionElement = driver.FindElement(By.ClassName(  
"ui-accordion ui-widget ui-helper-reset"));
```

Dies wird jedoch zu einer `InvalidOperationException` mit der Nachricht „Compound class names not permitted“ führen, da der Klassenname aus mehreren zusammengefügten Klassen besteht, die durch Leerzeichen voneinander getrennt sind. Aus diesem Grund muss der Cascading Style Sheets (CSS) Selektor genutzt werden.

Der Aufbau eines CSS Regelsatzes ist:

```
Selektor{  
    Eigenschaft: Wert;  
}
```

Als Zeichenkette kann ein CSS-Ausdruck mit der Notation

```
elementName[<einenschaftsName>='<wert>']
```

an die Funktion übergeben werden und beispielhaft folgenden Aufbau haben:

```
IWebElement accordionElement = driver.FindElement(By.CssSelector(  
"div[class = 'ui-accordion ui-widget ui-helper-reset']"));
```

Die dritte Option ist mittels der `role`. Hierbei muss auch der CSS-Selektor genutzt werden, da es keinen eigenen Positionsgeber hat. Es wird nach einem `<div>` Element geschaut, das ein `<role>` Element mit dem Wert 'tablist' enthält:

```
IWebElement accordionElement = driver.FindElement(By.CssSelector(  
"div[role = 'tablist'])");
```

Die CSS-Selektoren Aufrufe für einfache Selektoren können mit einer verkürzten Schreibweise aufgerufen werden. Einfache Selektoren umfassen den ID-, Klassen-, Attributs -, Typ- und Universalselektor.

Bei allen drei Optionen könnte auch der XPath verwendet werden, wodurch die Konten anhand ihres Pfads im Dokument lokalisiert werden. Dieser ist jedoch langsam und es wird empfohlen, die meist individuelle Id zu nutzen (vgl. Selenium, 1312). Aus diesem Grund wird, wenn vorhanden, die Id als Positionsgeber verwendet.

Wurde das Element gefunden, kann für dies die Methode `Click()` aufgerufen werden:

3 Automatisierte Tests

Tabelle 3.2: Einfache Selektoren (vgl. W3 Schools, 0101)

Selektor	Aufruf/Bsp	Beschreibung
Id	#id	selektiert das Element mit der Id = 'id'
Klassen	#class	selektiert das Element mit class='class'
Universal	*	Universalselektor, wählt alle Elemente aus
Element	element	selektiert alle Elemente dieses Typs
Klasse eines Elements	element.class	Wählt die Klasse des Elements
Mehrere Elemente	element1,element2,...	Wählt die Elemente

```
accordionElement.Click();
```

Hierbei wird ein einfacher Mausklick ausgeführt. Es ist auch möglich weitere Aktionen mit der Maus auszuführen, beispielsweise einen Doppelklick oder einen Rechtsklick auf ein Element (vgl. Selenium, 2112).



Abbildung 3.7: Dropdown des Kontaktformulars der S-Erleben-Website

Dadurch öffnet sich das Kontaktformular und weitere Elemente werden sichtbar, die identifiziert werden müssen, um mit ihnen interagieren zu können (siehe Abbildung 3.6). Die Input-Felder ‚Name‘, ‚Vorname‘, ‚E-Mail‘, ‚Betreff‘, ‚Telefonnummer‘ und ‚Nachricht‘ müssen ausgefüllt werden. Zudem muss bei der Dropdown-Liste eine Kategorie ausgewählt werden, wobei es hier nur die Auswahl ‚Allgemein‘ gibt (siehe Abbildung 3.7).

Die Input-Felder werden, wie das Akkordion-Element, mittels der `FindElement()`-Methode und der jeweiligen Id lokalisiert. Mit der `SendKeys()` Methode können Tastenkombinationen ausgeführt oder Zeichenketten als Parameter übergeben werden, die daraufhin in das Input-Feld geschrieben werden.

```
IWebElement nameInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_tbName"));
nameInput.SendKeys("Testerin");
```

```
IWebElement vornameInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_tbVorname"));
vornameInput.SendKeys("Selenium");

IWebElement eMailInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_tbEMail"));
eMailInput.SendKeys("lei@oevermann.de");

IWebElement betreffInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_Betreff"));
betreffInput.SendKeys("Test");

IWebElement telefonnummerInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_telefonnummer"));
telefonnummerInput.SendKeys("0123456789");

IWebElement nachrichtInput = driver.FindElement(By.Id(
    "main_content_Kontakt_updateKontakt_getKontakt_tbNachricht"));
nachrichtInput.SendKeys("Dies ist eine Testnachricht");
```

Die HTML Dropdown-Liste für die Kategorie hat zwei Kindelemente: die erste Option "Wählen Sie eine Kategorie" mit dem Wert -1, die den Default Wert repräsentiert und die zweite Option mit dem Wert "Allgemein":

```
<select id="main_content_Kontakt_updateKontakt_getKontakt_Kategorie"
name="main_content_Kontakt_updateKontakt_getKontakt_Kategorie"
class="radio">
    <option value="-1" selected="selected">Wählen Sie eine Kategorie
    </option>
    <option value="Allgemein">Allgemein</option>
</select>
```

Da es sich hierbei um ein **select** Element mit einer Id handelt, kann zuerst das Kontaktkategorie-Element mittels der Id lokalisiert und dann ein neues **SelectElement** erstellt werden, dem das Kontaktkategorie-Element als Parameter übergeben wird. Daraufhin kann die Methode **SelectByValue()** für das **SelectElement** mit dem Wert "**Allgemein**" als Parameter übergeben und aufgerufen werden, damit diese Kategorie ausgewählt wird:

```
IWebElement dropdown = driver.FindElement(
    By.Id("main_content_Kontakt_updateKontakt_getKontakt_Kategorie"));
SelectElement selectElement = new SelectElement(dropdown);
selectElement.SelectByValue("Allgemein");
```

3 Automatisierte Tests

The screenshot shows a web browser window for the S-Erleben website. The URL is erleben-ist-einfach.de/sk/kontakt. The page title is 'Kontakt'. The left sidebar has links for Kreissparkasse Köln, S-Mobilgeräteschutz, S-Fundservice, S-Cashback, Reisen, Shop, TicketService, Ratgeber, and Kontakt. Below these are social media icons for Facebook, Twitter, YouTube, and Instagram. The main content area has a red header with the S-Erleben logo and navigation links for Login, E-Mail Adresse, Passwort, Fresschen, and Passwort vergessen. The page title 'Ihre Nachricht an uns' is followed by a note: 'Gerne beantworten wir Ihre Fragen. Wir freuen uns auf Ihre Mitteilung.' A section titled 'Kontakt' contains an accordion menu with 'Allgemeine Fragen' expanded. Below it is a 'Kontaktformular' section with fields for Name*, Vorname*, E-Mail*, Betreff*, Telefonnummer für Rückfragen*, Kategorie*, and Nachricht*. The Nachricht field contains the text: 'Hallo, das ist ein automatisierter Oberflächentest für das Kontaktformular. Sie können diese Nachricht ignorieren.' A red 'Absenden' button is at the bottom right. A small note at the bottom left of the form says: '* Pflichtfeld'.

Abbildung 3.8: Ausgefülltes Kontaktformular der S-Erleben-Website

Das ausgefüllte Formular (siehe Abbildung 3.8) kann nun, durch Klicken des Senden-Buttons, abgeschickt werden:

```
<a class="red-btn1" tabindex="0"
   id="main_content_Kontakt_updateKontakt_getKontakt_btnErfassen"
   title="Anmelden" style="cursor:pointer;">Absenden</a>
```

Hierzu wird, wie zuvor beim Akkordeon, das Element mittels der Id lokalisiert und mit Click() geklickt. Der dazugehörige Aufruf lautet:

```
IWebElement sendenButton = driver.FindElement(By.Id("senden"));
sendenButton.Click();
```

3 Automatisierte Tests

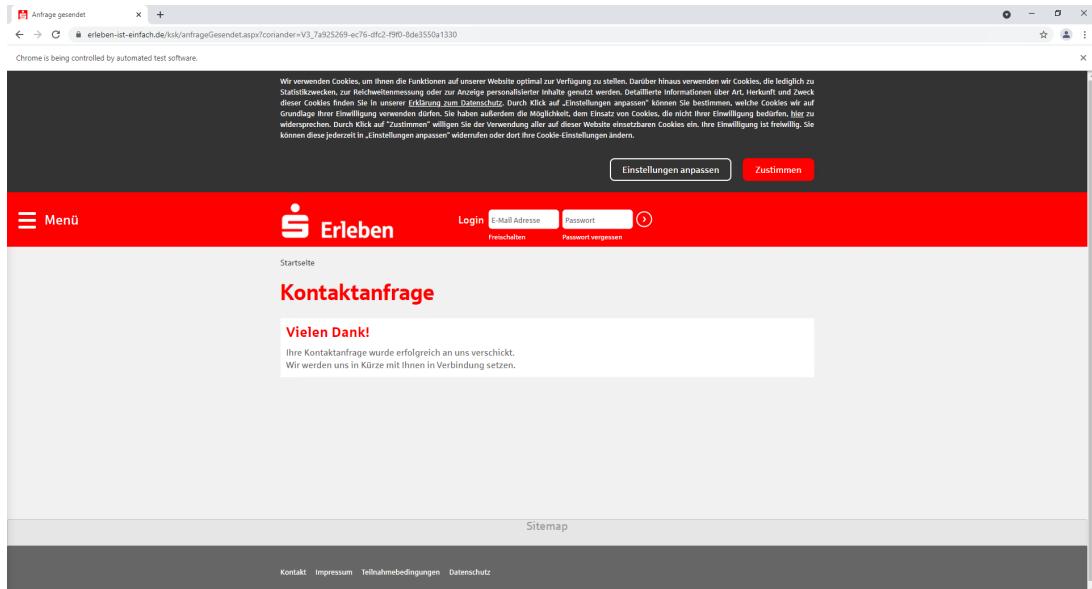


Abbildung 3.9: Seite mit Bestätigungstext nach Absenden des Kontaktformulars

Sollte dieser Schritt erfolgreich gewesen sein, wird eine neue Seite geladen, da es sich bei dem Button um einen Hyperlink handelt. Dies ist erkennbar an dem HTML Tag <a>. Auf dieser Seite wird ein Bestätigungstext angezeigt (siehe Abbildung 3.9)., Zuletzt muss der Chrome Driver mit der Methode `Quit()` geschlossen werden:

```
driver.Quit();
```

Hierbei werden der Browser, der Hintergrundprozess des Drivers und alle Fenster und Tabs der WebDriver-Session geschlossen. Wird die Methode nicht aufgerufen, laufen unter anderem Hintergrundprozesse weiter, die zu Fehlern führen können. (Selenium, 0812).

Vorläufiges Assert

Bevor der WebDriver geschlossen wird, kann beispielsweise überprüft werden, ob der entsprechende Bestätigungstext (siehe Abbildung 3.9), dessen HTML folgender ist:

```
<article class="box-white">
  <div class="box-white-inner">
    <h2>Vielen Dank!</h2>
    <p>Ihre Kontaktanfrage wurde erfolgreich an uns verschickt.<br>
      Wir werden uns in Kürze mit Ihnen in Verbindung setzen.</p>
  </div>
</article>
```

angezeigt wird.

Der Text eines `IWebElements` kann ausgelesen werden, wenn für das Element `.Text` aufgerufen wird. Bei diesem Aufruf wird eine Zeichenkette zurückgegeben. Für das zuvor beschriebene Element lautet diese:

```
"Vielen Dank!\r\nIhre Kontaktanfrage wurde erfolgreich an uns " +  
"verschickt.\r\nWir werden uns in Kürze mit Ihnen in Verbindung setzen."
```

Diese Zeichenkette kann in einer Variable gespeichert werden und mit der Methode `StartsWith()` kann überprüft werden, ob diese beispielsweise mit "Vielen Dank!" beginnt. Der Rückgabewert dieser Funktion ist ein Boolean und wird in einer neuen Variable gespeichert, damit der Wert bei der Fehlerbehebung überprüft werden kann:

```
var bestaetigungsText = driver.FindElement(By.CssSelector(  
    "[class = 'box-white-inner']]")).Text;  
var bestellungBestaetigt = bestaetigungsText.StartsWith("Vielen Dank!");
```

Das Überprüfen des Tests mit der `Assert`-Klasse wird ausführlicher im Abschnitt `Assert` 3.3.2 behandelt und die Ausführung von Tests im Abschnitt 3.6.

3.3 MSTest-Frameworks

Die Einbindung und Nutzung des zuvor erwähnten Test-Framework von Microsoft MS-Test (siehe Abbildung 3.1.2) wird in diesem Abschnitt beschrieben.

Zum Nutzen des Frameworks muss der entsprechende Namespace am Anfang der Testdatei eingebunden werden:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
```

Dieser Namespace enthält laut Microsoft (1611) unter anderem:

- Attribute zur Initialisierung und Bereinigung
- die Klasse `Assert`, um Bedingungen zu überprüfen
- die `TestContext` Klasse, bei der Informationen für den Test gespeichert und bereitgestellt werden

Auf diese wird in den folgenden Unterabschnitten eingegangen.

3.3.1 Attribute

Mit den Attributelementen für Tests wie `[TestClass()]` können Tests wartbarer und lesbarer gemacht werden (vgl. Osherove u. a., 2015, S. 43). Gleichzeitig werden so Testklassen und -methoden vom Framework erkannt und können ausgeführt werden. Dazu muss eine Testklasse sowohl das `[TestClass()]` als auch das `[TestMethod()]`-Attribut enthalten. Das `[TestClass()]` Attribut indiziert, dass die Klasse eine oder mehrere Testmethoden enthalten könnte, die mit dem Attribut `[TestMethod()]` gekennzeichnet sind (vgl. Microsoft, 1611). Wurde die Klasse nicht mit dem entsprechenden Attribut versehen, wird sie nicht als Testklasse gesehen und bei der Ausführung ignoriert.

```
[TestClass()]
public class TestKontaktformularKSK{

    [TestMethod()]
    public void TestKontaktformular(){
        //Testlogik
    }
}
```

Zusätzlich gibt es optionale Testattribute, die in der Tabelle 3.3 aufgeführt sind.

Tabelle 3.3: Weitere Testattribute (vgl. Microsoft, 1611)

Attribut	Beschreibung
<code>[ClassInitialize()]</code>	Wird verwendet, um Code auszuführen, bevor der erste Test in der Klasse durchgeführt wird.
<code>[ClassCleanup()]</code>	Wird verwendet, um Code auszuführen, nachdem alle Tests in einer Klasse gelaufen sind.
<code>[TestInitialize()]</code>	Wird verwendet, um Code auszuführen, bevor jeder Test ausgeführt wird.
<code>[TestCleanup()]</code>	Wird verwendet, um Code auszuführen, nachdem jeder Test ausgeführt wurde.

Nach dem Beenden eines Tests kann `[ClassCleanup()]` aufgerufen werden, um beispielsweise den Driver zu beenden. Es wird eine Funktion definiert, die `driver.Quit()` aufruft:

```
[ClassCleanup()]
private void TestBeenden(){
    driver.Quit();
}
```

Ohne eine Testumgebung, wie das MSTest-Framework, kann der Test auch in try-finally Blöcken geschrieben werden. Hierzu wird die Testlogik in den try-Block geschrieben und `driver.Quit()` in den finally-Block. Der Code des try-Blocks wird versucht auszuführen, während im finally-Block, auch im Fall einer Exception, Ressourcen des try-Blocks wieder freigegeben werden sollen (vgl. Microsoft, 2809).

```
try {
    //Testlogik
} finally {
    driver.Quit();
}
```

Alternativ zu den try-finally Blöcken, die verhindern, dass ausführbare WebDriver Dateien (.exe) offen bleiben, kann eine `using`-Anweisung für das Initialisieren des neuen IWebDrivers verwendet werden, da es sich hierbei um ein IDisposable-Element handelt und so auch die verwendeten Ressourcen wieder freigegeben werden (vgl. Microsoft, 2709):

```
using (IWebDriver driver = new ChromeDriver())
```

Beide Optionen verhindern, dass Instanzen des WebDrivers geöffnet bleiben und zu Fehlern führen.

3.3.2 Assert

Die Testlogik enthält meistens auch eine Assert Anweisung. Diese dient zur Prüfung, ob sich ein Test wie vorgesehen verhält. Dazu wird auf die `Assert` Klassen zugegriffen, welche vom `Microsoft.VisualStudio.TestTools.UnitTesting` Namespace bereitgestellt wird. Die Klasse `Assert` enthält unter anderem folgende Methoden mit unterschiedlichen Überladungen (vgl. Microsoft):

- `AreEqual()`
- `AreNotEqual()`
- `AreNotSame()`
- `AreSame()`
- `Equals()`
- `IsFalse()`
- `IsNotNull()`
- `IsNull()`
- `IsTrue()`
- `ThrowsException<T>()`

An die Anweisung `Assert.IsTrue(Boolean)` kann die Variable `bestellungBestaetigt` aus dem Abschnitt 3.2.2 übergeben und geprüft werden.

```
Assert.IsTrue(bestellungBestaetigt);
```

Sollte diese Variable nicht wahr sein, wird eine Exception geworfen und der Test gilt als nicht bestanden.

3.3.3 TestContext

Die `TestContext` Klasse wird genutzt, um Informationen für Tests bereitzustellen. Im Rahmen des Praxisprojekts wird die `TestContext` Klasse verwendet, um auf Testlaufparameter einer Laufzeiteinstellungsdatei (`.runsettings`) zuzugreifen. Dazu muss einerseits eine Laufzeiteinstellungsdatei erstellt und eingebunden werden (siehe Unterkapitel 3.4) und eine Instanz des `TestContext` in der Testklasse erstellt werden.

Auf die Parameter der Laufzeiteinstellungsdatei kann mittels der Eigenschaft `TestContext.Properties` zugegriffen werden (vgl. Microsoft, 1412). Im Folgenden wird auf den Testlaufparameter mit dem Namen "`kskUrlKontakt`" der `.runsettings` Datei (siehe Unterkapitel 3.4) zugegriffen, dessen Wert in einer Variable gespeichert und zu einer Zeichenkette konvertiert wird:

```
[TestClass()]
public class TestKontaktformularKSK{

    public TestContext TestContext { get; set; }

    [TestMethod()]
    public void TestKontaktformular(){

        var kskUrlKontakt =
            TestContext.Properties["kskUrlKontakt"].ToString();
        //Testlogik (Act + Assert)
    }
}
```

Diese Variable, die nun den Wert des Parameters enthält, kann im Test verwendet werden, um die URL aufzurufen:

```
driver.Navigate().GoToUrl(kskUrlKontakt);
```

Der Rest der Testmethode wird in den folgenden Kapiteln angepasst.

3.4 Runsettings

In diesem Unterkapitel wird die Dokumentation *Konfigurieren von Komponententests mithilfe einer RUNSETTINGS-Datei* von Microsoft (1412) genutzt.

Eine Laufzeiteinstellungsdatei wird für das Praxisprojekt verwendet, um, wie zuvor erwähnt (siehe Unterkapitel 3.3.3), für die Selenium Tests Laufzeitparameter anzulegen. Allgemein ist dies nicht unbedingt erforderlich, sondern nur, wenn Test konfiguriert werden sollen. Eine Laufzeiteinstellungsdatei ist eine XML-Datei, deren Erweiterung `.runsettings` ist. Sie können nach Bedarf im Menü von Visual Studio aktiviert oder deaktiviert werden. Für Tests aktiv verwendete Laufzeiteinstellungsdatei sind Testeinstellungsdateien.

Für die Selenium Tests der KSK wurden drei Laufzeiteinstellungsdateien erstellt:

- DEV

- QS
- live/ PROD

Dies liegt daran, dass Änderungen an der Website erst in der Entwicklungsumgebung (DEV) implementiert werden, dann in der Qualitätssicherungsumgebung (QS) geprüft werden und nach der erfolgreichen Prüfung in der Produktivumgebung (PROD) angewendet werden. Die KSK Website hat in den unterschiedlichen Umgebungen verschiedene URL und Zugangsdaten, die in den jeweiligen Dateien hinterlegt sind. Somit sind die Entwicklungsumgebungen voneinander getrennt.

Im Laufe des Projekts wurde jedoch auf die Laufzeiteinstellungsdateien für DEV und QS verzichtet, da aufgrund von häufigen Änderungen an DEV und QS und zu geringen Ressourcen nur noch live getestet werden sollte. Deshalb wird nur noch auf diese eingegangen.

3.4.1 Aufbau der Laufzeiteinstellungsdatei

Die Laufzeiteinstellungsdatei beginnt mit der Deklaration des XML-Dokuments, bei der die XML-Version und Angaben zur Zeichencodierung für den XML-Parser enthalten sind (vgl. Tim Bray u. a., 2008). Es wird die Version 1.0 und UTF-8 zur Encodierung verwendet.

Im RunSettings Element können Konfigurationselemente (`<RunConfiguration>`) definiert werden, wie beispielsweise `TestSessionTimeout`. Dies gibt die maximale Zeit in Millisekunden an, bis ein Test beendet wird.

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
    <!-- Konfigurationselemente -->
</RunSettings>
```

Neben dem `<RunConfiguration>` Elementen können folgende Konfigurationselemente in dem RunSettings Element definiert werden:

- `<DataCollectionRunSettings>`
- `<LoggerRunSettings>`
- `<MSTest>`
- `<TestRunParameters>`

Für das Praxisprojekt wird jedoch nur auf die `TestRunParameters` eingegangen. Sie enthalten einen Namen (name) und einen Wert (value) und werden genutzt, um Variablen zu verwalten, die entweder für mehrere Tests genutzt werden, wie der URL der KSK oder sich verändern könnten, wie Zugangsdaten und Telefonnummern. So können in mehreren Testmethoden Werte gleichzeitig geändert werden.

Hier ein Auszug der `TestRunParameters`:

```
<TestRunParameters>
    <!--URL-->
    <Parameter name = "kskUrl"
        value = "https://www.erleben-ist-einfach.de/ksk/">
    <Parameter name = "kskUrlKontakt"
        value = "https://www.erleben-ist-einfach.de/ksk/kontakt/">
</TestRunParameters>
```

Mittels der TestContext.Properties Eigenschaft wurde in dem Unterkapitel 3.3.3 bei spihaft auf den Wert des Parameters mit dem Namen "kskUrlKontakt" ge griffen.

3.5 Refactoring

In diesem Abschnitt wird der Test überarbeitet, ohne das Verhalten zu verändern.

3.5.1 Refaktoriertes Arrange

Nachdem die Testeinstellungsdatei eingebunden wurde, muss der Test überarbeitet werden. Für alle Variablen, die einen Parameter haben, muss auf dessen Eigenschaften zugegriffen werden. Zusätzlich wird eine Variable des Typs Boolean hinzugefügt, die definiert, ob das Kontaktformular abgesendet werden soll. So kann verhindert werden, dass beim Debuggen jedes Mal eine E-Mail versendet wird. Überarbeitet ist der Arrange-Teil dann folgender:

```
string name = TestContext.Properties["name"].ToString();
string vorname = TestContext.Properties["vorname"].ToString();
string eMail = TestContext.Properties["userEMail"].ToString();
string betreff = TestContext.Properties["betreff"].ToString();
string telefon = TestContext.Properties["telefon"].ToString();
string nachricht = TestContext.Properties["nachrichtKontaktformular"]
    .ToString();
string bestaetigungsText = TestContext
    .Properties["neuKontaktformularBestaetigung"].ToString();
bool kontaktformularSenden = Convert.ToBoolean(
    TestContext.Properties["kskKontaktformularSenden"]);
```

3.5.2 Refaktoriertes Act

Beim Ausführen der Tests können Fehler auftreten, wie eine NoSuchElementException. Dies kann geschehen, wenn eine Seite aufgerufen wird und ein Element der Seite gesucht wird, das noch nicht geladen wurde und somit auch noch nicht angezeigt wird. Dies liegt daran, dass der WebDriver eine blockierende API ist und die Webplattform asynchron ist. So kann vom WebDriver nicht der aktuelle Stand der DOM abgefragt werden. Dadurch entstehen unter anderem Wettlaufbedingungen (engl. Race Conditions).

„Eine Wettlaufsituation liegt dann vor, wenn mehrere nebenläufige Prozesse oder Threads auf denselben Daten arbeiten und das Ergebnis der parallelen Programmausführung von der jeweiligen Ausführungsreihenfolge im Einzelfall abhängt. Hieraus folgt

auch, dass mindestens einer der Threads schreibend auf die gemeinsam genutzten Daten zugreifen muss, damit eine Wettlaufsituation vorliegen kann. Wird ausschließlich lesend auf die gemeinsamen Daten zugegriffen, spielt die Reihenfolge dieser Zugriffe für das Ergebnis keine Rolle.“ (Hoffmann u. Lienhart, 2008, S. 67)

Im zuvor genannten Beispiel greifen also sowohl der WebDriver als auch die Seite gleichzeitig auf das Element zu, wobei der Browser das Element verändert und somit schreibend auf das Element zugreift.

Wenn die Seite noch nicht fertig geladen ist, liegt dies an der Strategie, die verwendet wird, um die Seite zu laden (engl. Page loading strategy). Diese wartet standardmäßig darauf, dass sich der Ladezustand (`document.readyState`) auf `complete` (dt. vollständig) setzt.

Um die Ladezeit zu verkürzen, kann die `eager` (dt. ungeduldige) Strategie verwendet werden. Hierbei werden zusätzliche Ressourcen wie Bilder nicht mit geladen. Wenn die Strategie zum Laden auf `none` (dt. keine) gesetzt wurde, wartet der WebDriver nur auf den Download der Seite (vgl Selenium, 0712c). Um Race Conditions zu vermeiden, kann auf das Element, welches gesucht wird, gewartet werden. Dazu muss das Paket `OpenQA.Selenium.Support.UI`; eingebunden werden.

Bei WebDriver wird zwischen explizitem Warten (engl. explicit wait), impliziertem Warten (engl. implicit wait) und ‚fließendem‘ Warten (engl. fluent wait) unterschieden. Für das Praxisprojekt wird nur die Strategie des expliziten und impliziten Wartes verwendet.

Explizites Warten

Beim expliziten Warten wird eine Zeitspanne und eine Bedingung definiert, die an eine Funktion weitergegeben werden. Dies führt dazu, dass der Test nicht weiterläuft, bis die Zeit abgelaufen ist oder die Bedingung erfüllt wurde (vgl Selenium, 0712d). Beispielsweise kann der WebDriver fünf Sekunden warten, bis das Akkordeonelement gefunden wird.

Um ein Wait mit einer Zeitspanne von fünf Sekunden zu initialisieren wird ein Objekt vom Typ `WebDriverWait` erstellt und diesem ein neues Objekt vom selben Typ zugewiesen und den zuvor erstellten WebDriver und die Zeit als Parameter übergeben.

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(5));
```

Für dieses Wait kann nun die Methode `Until<TResult>` aufgerufen werden. Dieser kann die Bedingung, dass das Element gefunden werden muss, als Funktionsreferenz übergeben werden und wird so lange ausgeführt bis die Funktion nicht mehr Null oder `false` zurückgibt, eine Exception geworfen wird, die nicht ignoriert werden soll oder die Zeit abgelaufen ist (vgl Sel).

```
IWebElement accordionElement = wait.Until(e =>
    e.FindElement(By.Id("accordion")));
```

Die `NoSuchElementException`, die vorher geworfen wurde, zählt zu den Exceptions, die ignoriert werden, bis die Zeitspanne von fünf Sekunden vorbei ist. Sollte die Zeitspanne vorbei sein und die Bedingung weiterhin `false` sein, wird ein `timeout error` geworfen.

Impliziertes Warten

Beim implizierten Warten fragt der WebDriver das DOM für eine bestimmte Zeit, bis es irgendein Element findet (vgl. Selenium, 0712d). Impliziertes Warten mit einer Zeitspanne von fünf Sekunden wird folgendermaßen initialisiert und ist dann für die gesamte Dauer einer Sitzung aktiviert.

```
WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(5));
```

Explizites und implizites Warten sollten nicht gleichzeitig verwendet werden, da es sonst zu erhöhten Wartezeiten kommen kann, die die Leistung verschlechtern (vgl. Selenium, 0712d).

Da Methoden wie `Click()` und `SendKeys()` keinen Rückgabewert haben sind sie synchron, anders als die `FindElement()` Methode.

Der überarbeitete Act Teil ist dann folgender und folgt unmittelbar auf den Code aus dem überarbeiteten Arrange Kapitel (3.5.1):

```
driver.Manage().Window.Maximize();
driver.Navigate().GoToUrl(kskUrlKontakt);

//Akkoordeon ausklappen
IWebElement accordionElement = wait.Until(e =>
    e.FindElement(By.Id("accordion")));

//Kontaktformular ausfüllen
IWebElement nameInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_tbName")));
nameInput.SendKeys(name);

IWebElement vornameInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_tbVorname")));
vornameInput.SendKeys(vorname);

IWebElement eMailInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_tbEMail")));
eMailInput.SendKeys(eMail);

IWebElement betreffInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_Betreff")));
betreffInput.SendKeys(betreff);

IWebElement telefonnummerInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_telefonnummer")));
telefonnummerInput.SendKeys(telefon);

IWebElement nachrichtInput = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_tbNachricht")));
```

```

nachrichtInput.SendKeys(nachricht);

//Auswählen der Kategorie
IWebElement dropdown = wait.Until(e => e.FindElement(By
    .Id("main_content_Kontakt_updateKontakt_getKontakt_Kategorie")));
SelectElement selectElement = new SelectElement(dropdown);
selectElement.SelectByValue("Allgemein");

//Senden des Kontaktformulars
if (kontaktformularSenden)
{
    IWebElement sendenButton = wait.Until(e =>
        e.FindElement(By.Id("senden")));
    sendenButton.Click();
}

```

Der Assert Teil wurde schon im Abschnitt 3.3.2 angepasst.

3.5.3 Auslagern von Methoden

Um die Testmethode übersichtlicher zu machen, können einige Befehle, wie beispielsweise das Ändern des Dropdown-Menüs als Methoden innerhalb der Klasse ausgelagert werden. Dies führt außerdem dazu, dass die Klasse mit ihren Methoden wartbarer wird und geschaut werden kann, ob diese neu erzeugten Methoden zukünftig von anderen Tests aufgerufen werden können. Dazu im Abschnitt 3.8.1 mehr.

Zum Auslagern des Codeabschnittes innerhalb der Klasse wird eine neue private Methode `DropdownAendern()` ohne Rückgabewert erstellt.

Dieser werden der WebDriver, ein Wait, eine Id als Zeichenkette und der Wert des zu wählenden Elements als Zeichenkette als Parameter übergeben.

```

private static void DropdownAendern(IWebDriver driver, WebDriverWait
                                    wait, string id, string value)
{
    IWebElement dropdown = wait.Until(e => e.FindElement(By.Id(id)));
    SelectElement selectElement = new SelectElement(dropdown);
    selectElement.SelectByValue(value);
}

```

Der Methodenaufruf lautet:

```
DropdownAendern(driver, wait,
"main_content_Kontakt_updateKontakt_getKontakt_Kategorie", "Allgemein");
```

Durch das Auslagern können weitere Dropdown-Menüs, die mittels einer Id lokalisiert werden, verändert werden.

3.6 Ausführen von Tests

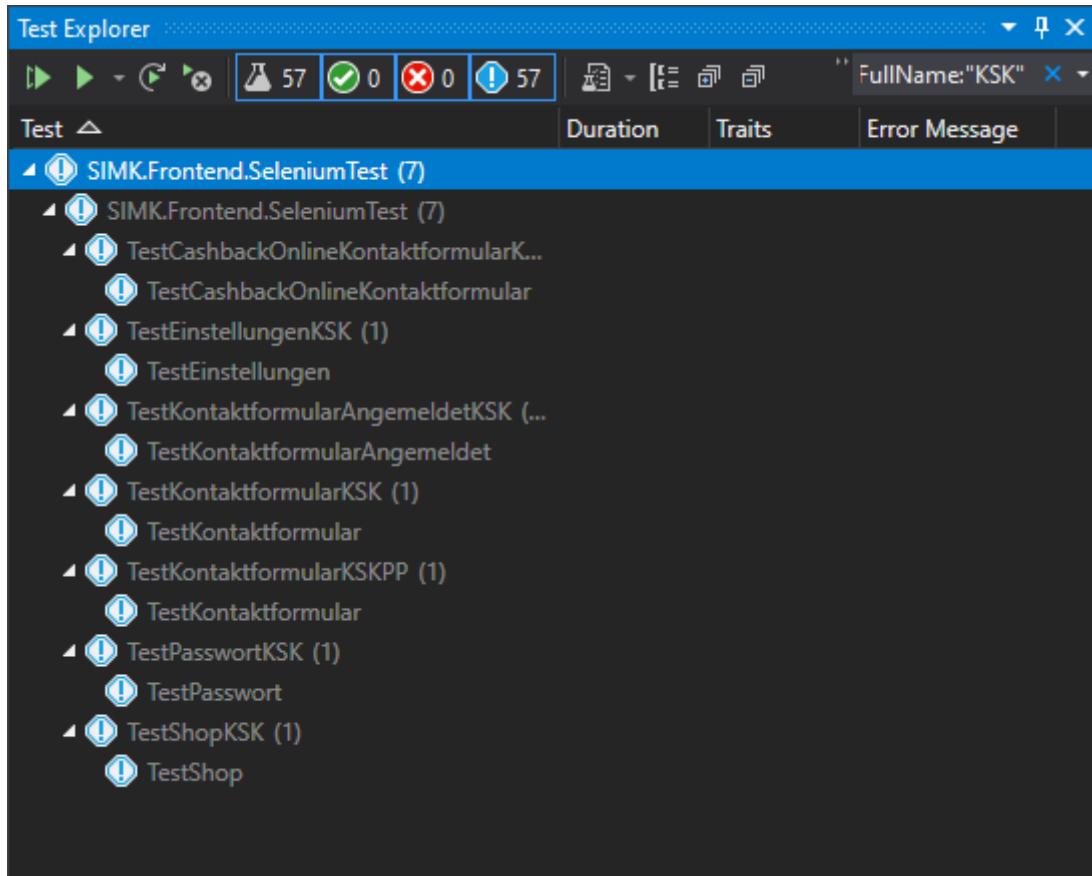


Abbildung 3.10: Test-Explorer mit den Testklassen und Testmethoden

Die Tests können mittels des Test-Explorers in Visual Studio ausgeführt werden. Der Test-Explorer gibt außerdem eine Übersicht über alle Tests, deren Dauer und den im Fehlerfall vorhandenen Fehlermeldungen (siehe Abbildung 3.10). Die Testergebnisse können im Detail mit einer Fehlermeldung angezeigt werden. Wenn die Tests erfolgreich sind, ändert sich das blaue Ausrufezeichen, das für einen nicht ausgeführten Test steht, zu einem grünen Haken bei einem erfolgreichen Durchlauf oder beim Fehlschlagen zu einem roten Kreuz.

Wird beispielsweise der Tests für das Ausfüllen des Kontaktformulars gestartet, braucht dieser nur wenige Sekunden, im Vergleich zum manuellen Durchlauf, der einige Minuten braucht.

3.7 Angepasstes Reporting

Beim Ausführen der Tests zeigten fehlgeschlagene Tests die Notwendigkeit genauerer Fehlermeldungen. Wie Fehler besser lokalisiert und behoben werden können, wird in

diesem Unterkapitel erläutert. Auf häufige Fehler und deren Behebung wird im Kapitel 4 eingegangen.

3.7.1 try-catch für Methoden

Für ausgelagerte Methoden, wie der `DropdownAendern()` aus dem Abschnitt 3.5.3, kann eine try-catch-Anweisung in der Methode genutzt werden.

Die Logik der Methode steht weiterhin im try-Block. Im catch-Block wird beim Werfen der Exception eine Nachricht mitgegeben, die beim Fehlschlagen der Methode in der Fehlermeldung mit angezeigt wird. Angepasst ist die `DropdownAendern()` Methode dann folgende:

```
private static void DropdownAendern(IWebDriver driver, WebDriverWait
                                    wait, string id, string value)
{
    try
    {
        IWebElement dropdown = wait.Until(e =>
            e.FindElement(By.Id(id)));
        SelectElement selectElement = new SelectElement(dropdown);
        selectElement.SelectByValue(value);
    }
    catch (Exception e)
    {
        TestMethoden.ScreenshotAufnehmen(driver, testContext);
        throw new Exception("Dropdown (" + id +
                            ") konnte nicht geändert werden.", e);
    }
}
```

Die Ausgabe einer Nachricht in einer Fehlermeldung ist in Abschnitt 4.3.1 zu sehen. Sie wird jedoch auch beim lokalen Debuggen im Test-Explorer und an der fehlgeschlagenen Codezeile angezeigt.

Wird dieses Schema für alle ausgelagerten Methoden verwendet, können auftretende Fehler schneller und einfacher analysiert werden.

Die Methode `ScreenshotAufnehmen()` wird im nächsten Abschnitt 3.7.2 behandelt.

3.7.2 Screenshot aufnehmen

Auf die Dokumentation *Working with windows and tabs* von Selenium (0812) wird sich in diesem Abschnitt bezogen.

Um nicht nur eine Nachricht zu erhalten, an welcher Stelle der Test fehlgeschlagen ist, sondern auch ein Bild, das dabei hilft, zu erkennen, ob es sich um einen Fehler des Tests oder der Website handelt, kann ein Screenshot erstellt werden.

Mit Selenium können Screenshots vom gesamten Browserausschnitt oder von einzelnen Elementen aufgenommen werden. Dazu muss ein neues Element des Typs `Screenshot` erstellt werden, dem der Rückgabewert der Methode `GetScreenshot()` zugewiesen

wird. Dieser ist vom Typ Screenshot und ist ein in Base64 kodierte Zeichenkette, die die Webseite repräsentiert. Der WebDriver muss als ITakesScreenshot Typ verwendet werden und deshalb konvertiert werden.

Der Screenshot kann dann gespeichert werden, indem ein zuvor generierter Dateiname mitgegeben wird, der aus dem Pfad und dem Datum mit Uhrzeit besteht. Diese Anweisungen werden ausgelagert in die private Funktion **ScreenshotAufnehmen()**, die eine Instanz des IWebDriver und des TestContext als Parameter übergeben bekommt und keinen Rückgabewert hat. Auch der Inhalt dieser Methode ist in einer Try-Catch Anweisung.

```
private static void ScreenshotAufnehmen(IWebDriver driver,
                                         TestContext testContext)
{
    try
    {
        string path = System.IO.Path.GetDirectoryName(
            System.Reflection.Assembly.GetExecutingAssembly().Location)
                    + "\\SeleniumTestScreenshots";
        System.IO.Directory.CreateDirectory(path);
        var fileName = path + "\\"
                        + DateTime.Now.ToString("dd.MM.yyyy-hh-mm-ss") + ".jpg";

        System.Threading.Thread.Sleep(5000);
        Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();
        ss.SaveAsFile(fileName);

        testContext.AddResultFile(fileName);
    }
    catch (Exception e)
    {
        throw new Exception("Screenshot hat nicht funktioniert", e);
    }
}
```

Bei allen Methoden, außer der **ScreenshotAufnehmen()** Methode, wird im Catch-Block die **ScreenshotAufnehmen()** Methode aufgerufen. Dadurch kann ein Screenshot für die Fehleranalyse verwendet werden. Sie helfen auch, Fehler zu kategorisieren, so ist ein temporärer Ausfall einer Seite, wie ein Serverfehler, deutlich an einem Bild zu erkennen. Ein Bildschirmfoto, das im Fehlerfall aufgenommen wurde, ist in Abschnitt 4.3.1 zu sehen.

3.8 Weitere Tests

Die weiteren Testfälle, welche im Abschnitt 3.2 aufgelistet sind, wurden auf eine sich gleichende Art implementiert. Auf einige Aspekte und Tests wird in diesem Unterkapitel eingegangen, jedoch nicht auf alle im Detail, da dies den Rahmen des Praxisprojekts überschreiten würde.

3.8.1 Testmethoden Klasse

Im Abschnitt 3.5.3 wurde die Methode `DropdownAendern()` innerhalb der Testklasse ausgelagert. Damit auch andere Testklassen auf diese und weitere Methoden zugreifen können, werden diese aus der Klasse ausgelagert. Dazu wurde eine neue Klasse mit dem Namen `TestMethoden` erstellt, die denselben Namespace hat, wie die Testklassen. In dieser befinden sich die öffentlichen Methoden.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Support.UI;

namespace SIMK.Frontend.SeleniumTest
{
    public class TestMethoden
    {
        //Methoden
    }
}
```

Die Methoden haben folgende Funktionen:

- Cookie-Abfrage bestätigen
- Dropdown ändern mit einem SelectElement
- Klicken eines Elements, das eine Id hat
- Text an ein Element senden, das eine Id hat
- URL aufrufen
- User einloggen
- Zum Menüpunkt navigieren

Die Methode `DropdownAendern()` wurde mit einer Anpassung des Selektorauftrufs in die `TestMethoden` Klasse übernommen, damit sie nicht nur für Dropdown-Menüs, die mittels einer Id lokalisiert werden können, funktioniert, sondern auch mit weiteren CSS-Selektoren:

```
private static void DropdownAendern(IWebDriver driver,
                                     WebDriverWait wait, string selector, string value)
{
    try
    {
        IWebElement dropdown = wait.Until(e =>
```

3 Automatisierte Tests

```
    e.FindElement(By.CssSelector(selector)));
    SelectElement selectElement = new SelectElement(dropdown);
    selectElement.SelectByValue(value);
}
catch (Exception e)
{
    TestMethoden.ScreenshotAufnehmen(driver, testContext);
    throw new Exception("Dropdown (" + id +
        ") konnte nicht geaendert werden.", e);
}
}
```

Auf die Methoden kann aus den Testklassen zugegriffen werden. So beispielsweise für die Methode `ScreenshotAufnehmen()`:

```
TestMethoden.ScreenshotAufnehmen(driver, testContext);
```

Dies verringert Coderepetitionen, da Codeabschnitte wiederverwendet werden. Des Weiteren sind Methoden einfacher zu warten, was die Codequalität erhöht. Weitere Aufrufe finden sich im Abschnitt 3.8.2.

3.8.2 Beispiel Shop-Test

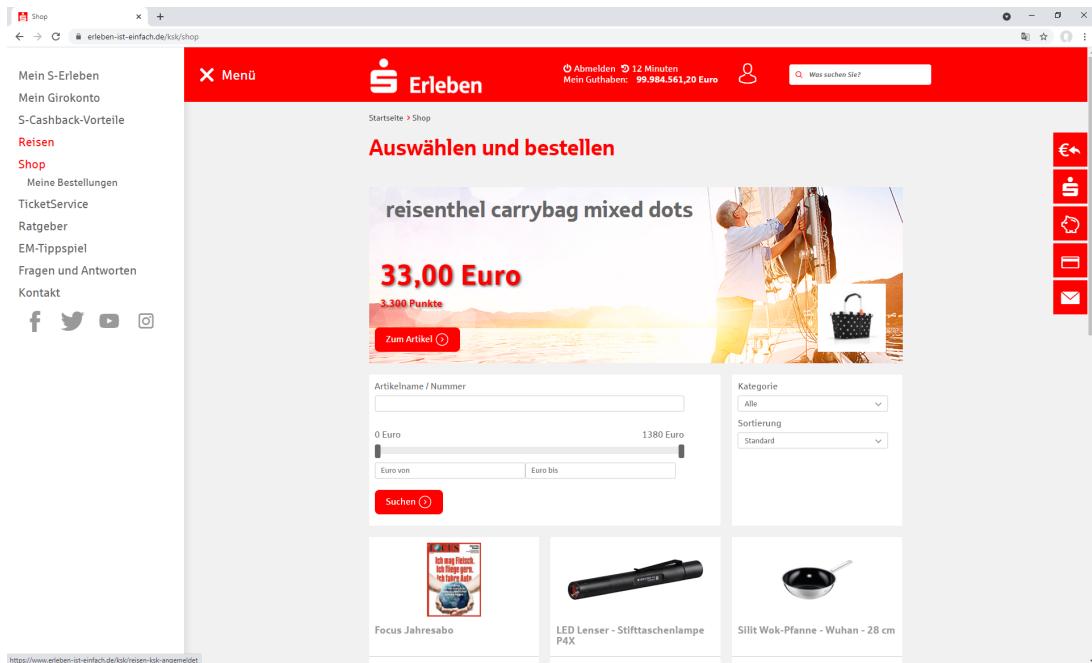


Abbildung 3.11: Shop der S-Erleben-Website

Die KSK Website enthält einen Shop. Um diesen mit den wichtigsten Funktionen zu testen, müssen mehrere Fälle abgedeckt werden. Folgende Aspekte werden in der Testmethode `TestShop()` getestet und sind in Abbildung 3.11 zu sehen:

- einen zufälligen Artikel bestellen
- mehrere Artikel bestellen
- eine Artikelseite aufrufen
- Suchfunktion für Artikel
- Preisspanne in der Artikelübersicht
- Menge auf der Artikelseite ändern
- Menge im Warenkorb ändern
- Artikel aus dem Warenkorb löschen
- Gutscheincode einlösen im Warenkorb
- Cashback ändern im Warenkorb
- Rechnungsanschrift ändern im Bestellprozess

Da nicht bei jedem Testdurchlauf jede Funktion getestet werden muss, wurden für jeden Aspekt ein **TestRunParameter** in der .runsettings Datei angelegt, die den Wert **true** oder **false** haben. Variablen des Typs Boolean werden in der Testmethode **TestShop()** angelegt, diesen wird mit einer Typkonvertierung der entsprechende Parameter der .runsettings Datei zugewiesen. Mittels **if**-Anweisungen, die diese Variablen prüfen, werden dann die entsprechenden Codeabschnitte ausgeführt.

Damit dem Warenkorb mehrere Artikel hinzugefügt werden, wird eine **for**-Schleife verwendet. In dieser wird der Driver jedes Mal wieder zum Shop navigiert und Artikel werden dem Warenkorb hinzugefügt, bevor Änderungen am Warenkorb vorgenommen werden oder die Bestellung abgeschickt wird.

```
[TestMethod()]
public void TestShop()
{
    using (IWebDriver driver = new ChromeDriver())
    {
        try
        {
            //Arange Teil

            driver.Manage().Window.Maximize();

            //Seite der KSK aufrufen
            TestMethoden.UrlAufrufen(driver, w, kskUrl, TestContext);

            //Cookie Abfrage
            TestMethoden.CookieAbfrageBestaetigen(
                driver, wait, TestContext);
        }
    }
}
```

3 Automatisierte Tests

```
//User Daten + PW eingeben und bestaetigen
TestMethoden.UserEinloggen(driver, wait, userEMail,
    userPasswortTestContext, true, kskUrl);

for (int i = 0; i < 5; i++) {

    driver.Navigate().GoToUrl(
        "https://www.erleben-ist-einfach.de/kskshop");
    //Artikelname suchen
    if (artikelNameSuchen)
    {
        ArtikelnameSuchen(driver, wait, artikelName,
            TestContext);

        //Preisspanne von bis suchen
        if (preisspanneSuchen)
        {
            PreisspanneSuchen(driver, wait, preisVon, preisBis,
                TestContext);

            //suchen, wenn mindestens eins true
            if (artikelNameSuchen || preisspanneSuchen)
            {
                SuchenClicken(driver, wait, TestContext);

                //Ware auswaehlen
                WareAuswaehlen(driver, wait, rnd, artiekseiteAufrufen,
                    TestContext);
                //menge aendern aufProduktseite (vor Warenkorb)
                if (artiekseiteAufrufen)
                {
                    if (warenkorbAnzAendern)
                    {
                        ProduktMengeAendern(driver, wait, selectElement,
                            Anzahl, TestContext);
                    }
                    WarenkorbAufArtikelseiteClicken(driver, wait,
                        TestContext);

                    //Menge im Warenkorb aendern
                    if (warenkorbMengeAendern)
                    {
                        WarenkorbMengeAendern(driver, wait, anzWarenkorb,
                            TestContext);
                    }
                }
            }
        }
    }
}
```

3 Automatisierte Tests

```
//artikel aus warenkorb loeschen
if (artikelAusWarenkorbLoeschen)
{
    ArtikelAusWarenkorbLoeschen(driver, wait, TestContext);

//cashback
if (cashbackAendern)
{
    CashbackAendern(driver, wait30, rnd, TestContext);
}
if (gutscheinTesten)
{
    gutscheincodeEingeben(driver, wait, gutscheincode,
        TestContext);
}
//bestellen
bestellenClicken(driver, wait, TestContext);

//rechnungsanschrift ändern
if (rechnungsanschriftAendern)
{
    RechnungsanschriftAendern(driver, wait, telefonnummer,
        TestContext);
}
//agb akzeptieren
AgbAkzeptieren(driver, wait, TestContext);
//weiter (bestellvorgang)
WeiterBestellvorganClicken(driver, wait, TestContext);
//bestellung abschliessen
BestellungAbschliessen(driver, wait, TestContext)
Assert.IsTrue(BestellungUeberpruefen(driver, wait,
    bestaetigungsText, TestContext));
}
finally
{
    driver.Quit();
}
}
```

Dies hat den Vorteil, dass der Test nur einmal laufen muss und Entwickler:innen die entsprechenden Bedingungen selbst in der .runsettings Datei einstellen können, ohne den Code des Tests ändern zu müssen.

4 Einbindung in das bestehende Projekt

In diesem Kapitel werden die Tests in das bestehende Projekt bei Azure DevOps eingebunden. Azure DevOps wird kurz erklärt und danach, wie die Tests eingebunden sind und wie die Testresultate überprüft werden können.

4.1 Azure DevOps

Um Tests in einen Release einzubinden und Berichte über den Status von den Testfällen zu erhalten, werden die Tests in Azure DevOps einbezogen. Azure DevOps ist ein Service von Microsoft, mit dem kollaborativ Softwareprojekte erstellt, geplant und verwaltet werden können.

4.2 Einbindung in einen Release



Abbildung 4.1: Ein Release der SIMK Projektpipeline mit den unterschiedlichen Stages

Eine Release Pipeline kann aus mehreren Stages (dt. Phasen) bestehen, die vom Entwicklungsteam festgelegt werden. Die aktuellen Stages entsprechen den Umgebungen DEV, QS und LIVE. Für die Selenium Tests wurden zwei Stages mit der Bezeichnung 'UI-Tests' (User Interface - Benutzerschnittstelle) erstellt (siehe 4.1). Die eine wird im Zusammenhang mit der Stage DEV ausgeführt, die andere mit der LIVE Stage.

Wenn eine Stage gelaufen ist und erfolgreich war, wie zum Beispiel LIVE, dann hat diese eine grünes Häkchen (succeeded), ansonsten ein rotes Kreuz (failed). Lief eine Stage nicht (not deployed), bleibt sie unverändert.

4.3 Überprüfung der Testresultate

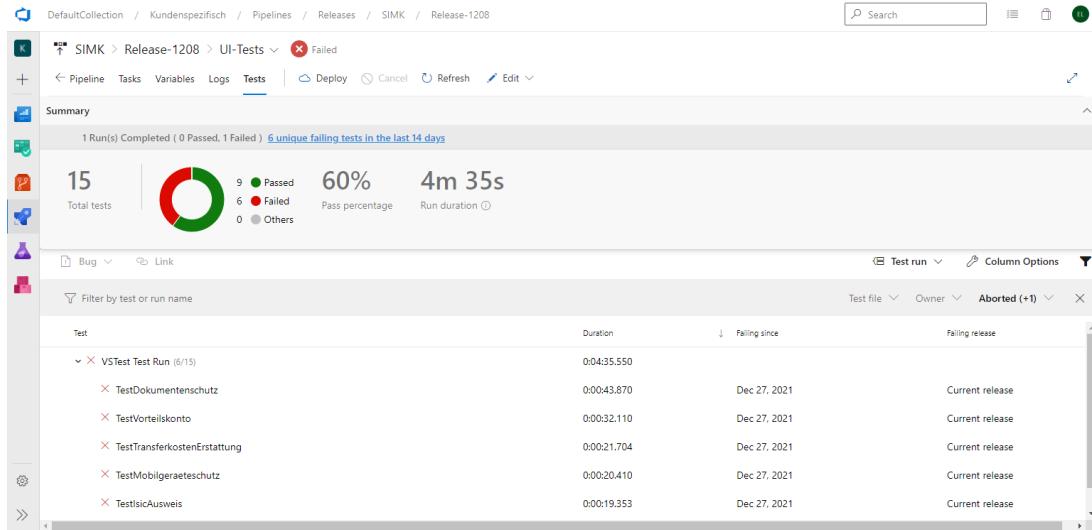


Abbildung 4.2: Testübersicht einer Stage in Azure DevOps

Für jeden Release können die Testresultate überprüft werden. Dazu wird die Stage **UI-Tests** geöffnet. Diese enthält eine Übersicht und eine Auflistung der fehlgeschlagenen Tests. Die Übersicht gibt Aufschluss über die Anzahl der ausgeführten Tests, deren Erfolgsquote und die Dauer der Laufzeit (siehe Abbildung 4.2).

4.3.1 Detaillierte Testresultate

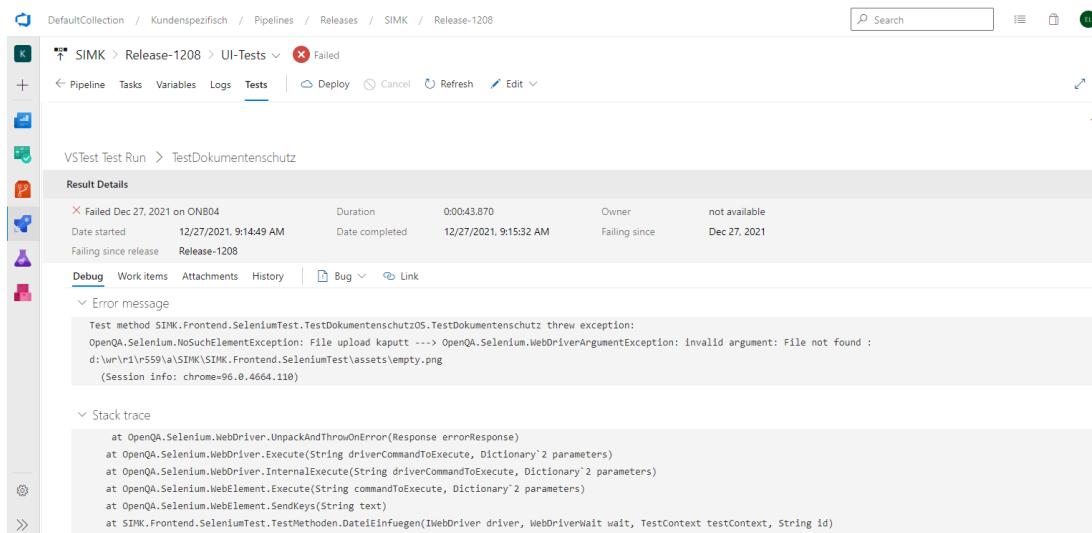


Abbildung 4.3: Detaillierte Testresultate: Debug

4 Einbindung in das bestehende Projekt

Ist ein Test fehlgeschlagen, können die detaillierten Testresultate angeschaut werden, in dem der Test geöffnet wird. Die Registerkarte **Debug** beinhaltet unter anderem die Fehlermeldung (engl. error message). Diese enthält die Information, welche Exception geworfen wurde und zusätzlich die mitgegebene Zeichenkette, der fehlgeschlagen Funktion aus dem catch-Block (siehe Abbildung 4.3).

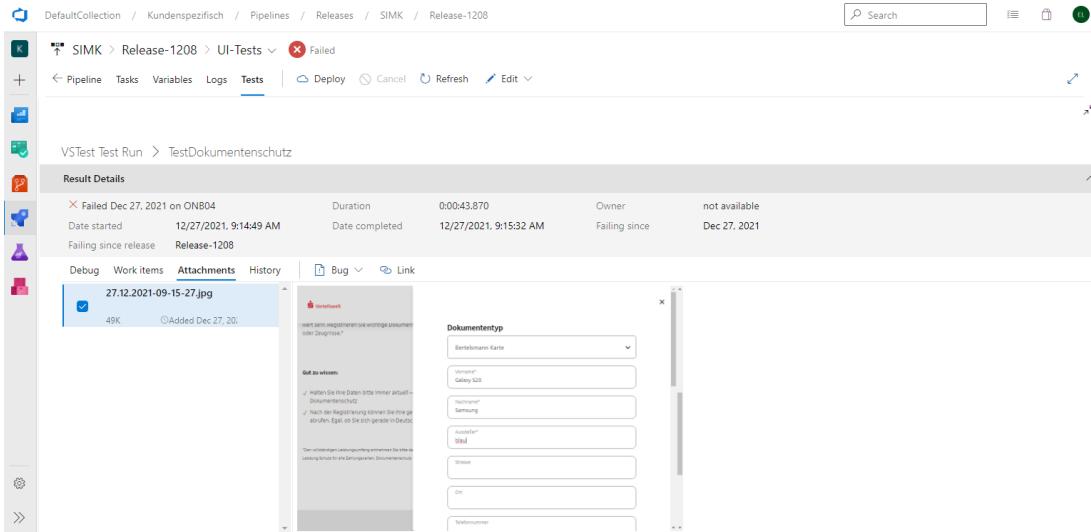


Abbildung 4.4: Detaillierte Testresultate: Attachments

Da der Catch-Block aller Methoden, abgesehen von der `TakeScreenshot()` Methode (siehe Abschnitt 3.7.2) selbst, diese aufruft, wurde im Fehlerfall auch ein Screenshot erstellt. Dieses befindet sich unter der Registerkarte **Attachments** (dt. Anhänge) (siehe Abbildung 4.4) und zeigt die Webseite an.

Sowohl die mitgegebene Nachricht im Fehlerfall als auch der Screenshot erleichtern die Lokalisierung eines Fehlers erheblich. Dadurch kann geprüft oder auch direkt gesehen werden, ob es sich um ein Fehler des Tests oder der Website handelt.

4.4 Auftretende Fehler

Dieser Abschnitt zeigt Fehler, die bei unterschiedlichen Testdurchläufen entdeckt wurden. Es wird beschrieben, was die Gründe für die Fehler sind und wie diese behoben werden können. Einige dieser Fehler wurde auch schon im Kapitel 3 erwähnt.

4.4.1 Session not created

Der `session not created` ist eine `System.InvalidOperationException`. Die Nachricht der Fehlermeldung zeigt dies und dass der Grund für den Fehler die unterschiedlichen Versionen des lokalen Chrome-Browsers und des ChromeDrivers sind:

Message:

```
Test method SIMK.Frontend.SeleniumTest.TestShopKSK.TestShop threw exception:
```

```
System.InvalidOperationException: session not created:  
This version of ChromeDriver only supports Chrome version 97  
Current browser version is 96.0.4664.110 with binary path  
C:\Program Files\Google\Chrome\Application\chrome.exe  
(SessionNotCreated)
```

Dieser Fehler führt dazu, dass keine Sitzung erstellt werden kann.

Um ihn zu beheben, muss der Chrome-Browser auf die Version des ChromeDrivers aktualisiert werden. Danach wird der ChromeDriver wieder gestartet und Tests können durchgeführt werden. Zur Vermeidung dieses Problems können unter anderem Pakete von Drittanbietern verwendet werden, wie beispielsweise das `WebDriverManager.Net` von `rosolko` auf GitHub, welches automatisch die Binärdateien des WebDrivers innerhalb eines .NET Projekts verwaltet.

Für einige Tests wurde geschaut, ob die Hyperlinks des Menüs (siehe Abbildung 1.1) funktionieren, da die Benutzer:innen meistens nicht direkt die URL der Webseite, die sie besuchen wollen, kennen und eintippen, sondern über das Menü dorthin gelangen. Das Menü wird jedoch nur angezeigt, wenn das Browserfenster groß genug ist, ansonsten ist das Menü eingeklappt und kann über das Hamburger-Menü-Icon geöffnet werden. Ist das Menü eingeklappt und der Tests soll einen Menüpunkt klicken, führt dies zu einer `NoSuchElementException`. Verhindert werden kann dies durch das Maximieren des Browserfensters (siehe Abschnitt 3.2) und durch eine Anpassung des Tests, bei der abgefragt wird, ob das Menü angezeigt wird:

```
private static void OpenMenu(WebDriverWait wait)  
{  
    var menuDisplayed = ElementHasClass(wait.Until(e  
        => e.FindElement(By.TagName("body"))), "open");  
    if (!menuDisplayed)  
    {  
        wait.Until(e => e.FindElement(  
            By.CssSelector("[class = 'menubtn']"))).Click();  
    }  
}
```

So werden für alle Tests dieselben Bedienungen geschaffen und das Menü ist immer offen.

4.4.2 No such element

Bei einer `NoSuchElementException` kann ein Element nicht lokalisiert werden. Dies kann unterschiedliche Gründe haben, einer davon ist, dass sich der Locator des Elements geändert hat. Sollte ein Element keinen eindeutigen Locator haben, wie eine für die Seite einmalige Id, kann der XPath verwendet werden. Dieser kann beim Untersuchen der zu testenden Webseite direkt kopiert werden. Der XPath identifiziert Elemente anhand ihrer Position in der DOM.

Das Problem an der Verwendung von XPath Selektoren für Websites, an denen noch Änderungen vorgenommen werden, ist, dass sich durch die Änderungen meist auch der

XPath-Ausdruck verändert und so das Element nicht mehr gefunden werden kann. Wie beispielsweise in dieser Nachricht einer Fehlermeldung eines sich in der Entwicklung befindenden Tests:

```
Test method
SIMK.Frontend.SeleniumTest.TestVorteilskontoOS.TestVorteilskonto
threw exception:
OpenQA.Selenium.NoSuchElementException:
Vorteilskontoauszahlung hat nicht funktioniert.
---> OpenQA.Selenium.WebDriverTimeoutException:
Timed out after 10 seconds
---> OpenQA.Selenium.NoSuchElementException: no such element:
Unable to locate element:
{"method":"xpath","selector": "//*[@id='mainForm']/div[3]/main/section
/div/div/div/div[1]/div[2]/div[2]/div[2]/div[2]/div
/simk-erstattung-liste/div/span/p/a"}
(Session info: chrome=97.0.4692.71)
```

Der Fehler kann behoben werden, indem der XPath des Elements aktualisiert wird, der partielle XPath verwendet wird oder versucht wird, das Element auf eine andere Weise eindeutig zu lokalisieren. Verhindert werden kann der Fehler jedoch durch die Zuweisung eines eindeutigen Locators, der dann für den Test verwendet werden kann. Dazu muss ein Ticket mit der Beschreibung des Elements und einer Bitte zur Erstellung einer Id erstellt werden. Dieses wird dann einem Teammitglied zugeordnet, das die Id dann im Code ergänzt. Gleichzeitig könnte mittels des Ticketsystems auch eine Mitteilung über die Änderung einer getesteten Webseite erstellt werden, wodurch Tests überprüft und gegebenenfalls angepasst werden können. Dies zeigt, dass die Entwicklungsaufgaben deutlich von den Tests getrennt werden und die Tests unabhängig von der Entwicklung sind, was Qualitätsanforderungen der ISO 27001 (siehe Abschnitt 1.3) sind.

Weiter Gründe für eine `NoSuchElementException` können sein, dass das Element entweder nicht mehr existiert oder es nicht geladen wurde. Sollte das Element nicht mehr existieren, muss der Test angepasst werden. Lädt die Seite jedoch nicht schnell genug, um ein Element zu lokalisieren, kann ein `wait` verwendet werden (siehe dazu Kapitel 3.5.2).

4.4.3 Sicherheitszertifikat abgelaufen

Beim Aufrufen einer Website sorgt ein Sicherheitszertifikat für die Verschlüsselung der Datenübertragung. Ist dies nicht aktuell, kann nicht sicher auf die Seite zugegriffen werden.

Dass das Sicherheitszertifikat für die Website abgelaufen ist, kam zwar nur einmal vor, jedoch ist es auch wichtig, dass dieser Fehler schnell behoben wird, damit die Website weiterhin genutzt werden kann. Dieses Problem fällt direkt nach dem Aufruf der URL auf, da dann keins der eigentlichen Elemente vorhanden ist.

Der Fehler konnte behoben werden, indem die Website ein neues Sicherheitszertifikat bekommen hat.

4.4.4 Testkonten verfügen nicht über ausreichende Berechtigungen

Für einige Testkonten waren die Berechtigungen nicht ausreichend, sodass beispielsweise das Passwort nicht geändert werden konnte. Für diesen Fall wurde ein anderes Testkonto angelegt, dem diese Berechtigung erteilt wurde. So kann auch verhindert werden, dass der Zugang zum Konto gesperrt wird, wenn der Test zum Passwort ändern fehlgeschlagen ist.

Neben der nicht vorhandenen Berechtigung für das Ändern des Passwortes, kann es auch das Problem geben, dass zu wenig Guthaben auf dem Testkonto vorhanden ist und Bestellungen nicht abgeschlossen werden können. Aus diesem Grund wurde für das Konto ein Gutscheincode erstellt, wodurch dem Konto ein ausreichender Betrag zur Verfügung gestellt wurde.

4.4.5 Kontaktformulare werden nicht gesendet

Im Falle des Tests für das Cashback-Online-Kontaktformular wurde häufiger das ausgefüllte Formular nicht abgesendet, weshalb die Überprüfung des Bestätigungstexts fehlgeschlagen ist. Bei der Betrachtung des Screenshots war das Formular vollständig ausgefüllt und der Button zum Absenden war auch vorhanden.

Bei der manuellen Überprüfung und der Auswertung des Errorlogs fiel auf, dass das Formular nicht abgesendet werden kann, wenn beim autocomplete Dropdown für den Partner, bei dem eingekauft wurde, nur der Name eingegeben wurde, jedoch nicht noch der entsprechende Partner aus der Liste ausgewählt wurde. Dieser Fehler fiel beim automatisierten Testen nicht auf, da die autocomplete Vorschläge nicht mehr zu sehen sind, wenn andere Formularfelder ausgefüllt werden und da auf der Webseite keine Fehlermeldung angezeigt wurde. Diese Fehlermeldung ist auch für Benutzer:innen elementar, da diese nicht die Fehlermeldung für die Seite aus dem Errorlog lesen können.

4.4.6 Client- und Serverfehler

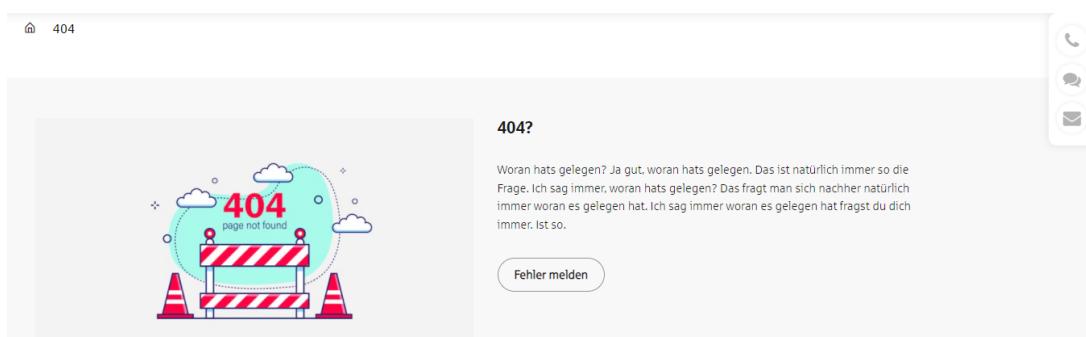


Abbildung 4.5: Clientfehler: 404

Im Laufe der Testentwicklung und -durchführung wurden Client- und Serverfehler gefunden. Unter anderem der Client-Fehler mit dem Statuscode 404 (siehe Abbildung 4.5), der definiert, dass die angeforderte Ressource nicht gefunden wurde. Dieser wird

4 Einbindung in das bestehende Projekt

beispielsweise angezeigt, wenn mittels eines Buttons auf eine neue Seite navigiert werden soll und diese nicht verlinkt ist. Dieser Fehler kann behoben werden, indem die Verlinkung korrigiert wird.

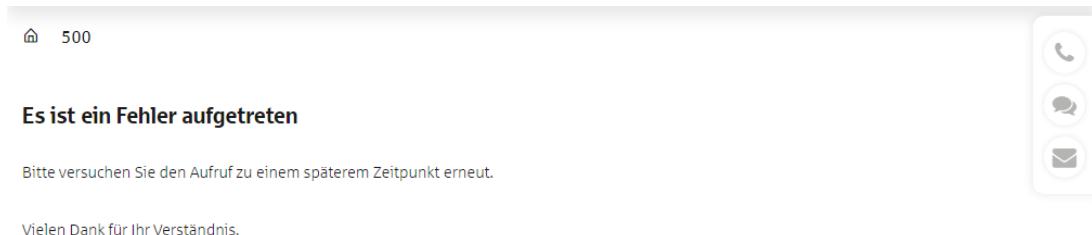


Abbildung 4.6: Serverfehler: 500

Serverfehler können entstehen, wenn zum Beispiel der Server ausfällt oder Fehler bei der Entwicklung entstehen. Dies führt jedoch dazu, dass die gesamte Website ausfällt, wodurch bei einem Serverfehler kein Test mehr erfolgreich ist (siehe Abbildung 4.6).

4.4.7 Darstellungsfehler

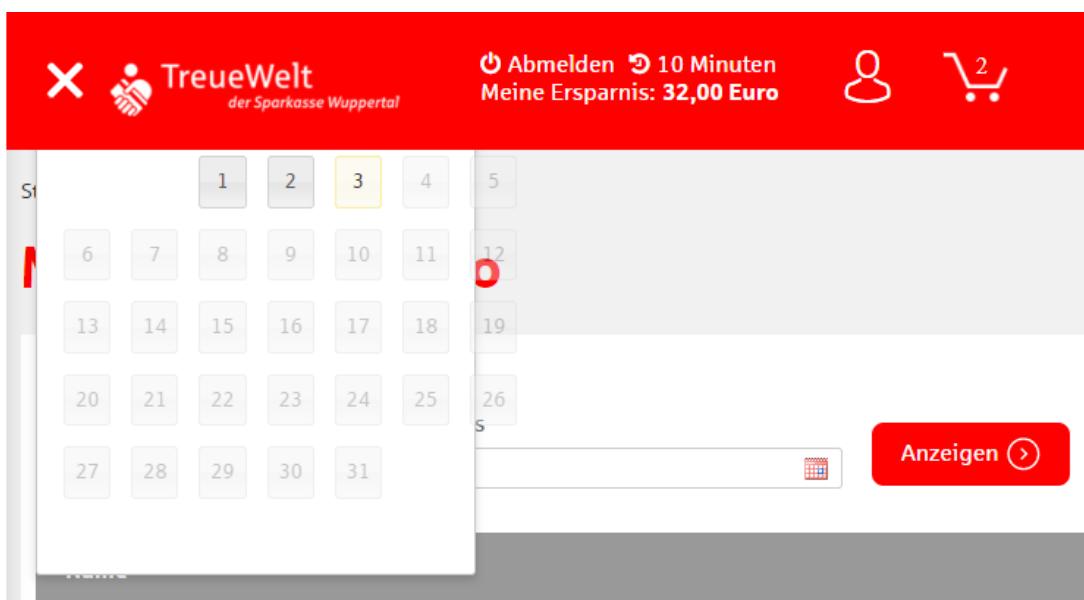


Abbildung 4.7: Fehlerhafte Darstellung eines Kalenders

Bei der Entwicklung der Tests wird sich intensiv mit dem User Interface (UI) der Website beschäftigt. Hierbei fallen häufig Fehler in der Darstellung auf, wie beispielsweise bei dem Kalender aus Abbildung 4.7. Für diese Art von Fehlern wird ein Ticket für das Frontend-Team erstellt, die diesen Fehler dann beheben und somit für eine nutzerfreundlicheres UI sorgen.

5 Fazit und Aussicht

Dieses Kapitel zieht ein Fazit, gibt Aussichten für den weiteren Verlauf des Projekts und für Themen eines Praxisprojekts oder einer Bachelorarbeit, die auf diesem Praxisprojekt aufbauen können.

5.1 Fazit

Die Tests erfüllen mit der verwendeten Software die projektspezifischen Anforderungen (siehe Abschnitt 1.4). Gleichzeitig werden auch alle testspezifischen Qualitätsanforderungen der ISO 27001 (siehe Abschnitt 1.3), bis auf die Trennung der Entwicklungs-, Tests und Betriebsumgebung, erfüllt. Diese Anforderung für zukünftige Tests zu erfüllen ist ein operatives Ziel (siehe Abschnitt 5.2.3).

Für alle in der Testplanung festgelegten Bereiche wurden Tests entwickelt, die kontinuierlich geprüft, weiterentwickelt und gepflegt werden.

Der erste Testdurchlauf war zu 60 Prozent erfolgreich. Es sind also bei zwei von fünf Tests Fehler aufgetreten, die an der Implementierung der Tests lagen. Im weiteren Verlauf waren durchschnittlich 90 Prozent erfolgreich bei einem bis zwei Testläufen pro Woche.

Da fehlgeschlagene Tests nicht bedeuten müssen, dass diese fehlerhaft sind, sondern das Ziel, Fehler auf der Website zu finden, erfüllen, sorgen diese für eine bessere Qualität der Website und somit auch für eine erhöhte Zufriedenheit der Nutzer:innen.

Durch das Erstellen der verschiedenen Tests wurden schon viele Fehler bei der Implementierung der Tests entdeckt, da sich hierbei intensiv mit der Website auseinandergesetzt werden muss. Weitere Fehler werden durch die laufenden Tests schneller erkannt und so kann beispielsweise Ausfällen der Website vorgebeugt werden. Gleichzeitig verringern die automatisierten Tests den Aufwand der Entwickler:innen in Bezug auf das manuelle Testen.

Es werden regelmäßig Fehler gefunden, die ohne automatisierte Test nicht oder erst spät aufgefallen wären (siehe Abschnitt 4.4). Jedoch muss beachtet werden, dass nicht der Test verändert werden sollte, wenn sich ein Element nicht so verhält, wie beabsichtigt, sondern der Fehler der Website behoben werden muss.

Im Rahmen des Praxisprojekts wurden nur ausgewählte Bereiche und Funktionen der KSK Website getestet. Aufgrund des Umfangs der Website konnten also nicht alle Bereiche und Funktionen getestet werden. Beim Erstellen von automatisierten Tests muss überprüft werden, welche die wichtigsten und fehleranfälligsten oder aufwändigsten Abläufe und Funktionen einer Website sind und nach diesen priorisiert werden.

5 Fazit und Aussicht

Wie häufig soll getestet werden?

<input checked="" type="radio"/> Einmalig	<input type="radio"/> Quartalsweise	<input type="radio"/> Monatlich
---	-------------------------------------	---------------------------------

Wie komplex ist die App bzw. Webseite?

<input type="radio"/> Klein	<input checked="" type="radio"/> Mittel	<input type="radio"/> Groß
-----------------------------	---	----------------------------

Auf welchen Plattformen?

 iOS	 Android	 Windows Phone	 Web
<input type="checkbox"/> iPhone	<input type="checkbox"/> Android Phone		<input type="checkbox"/> Desktop Browser
<input type="checkbox"/> iPad	<input type="checkbox"/> Android Tablet		<input type="checkbox"/> Phone Browser
<input type="checkbox"/> Apple Watch	<input type="checkbox"/> Android Watch		<input type="checkbox"/> Tablet Browser
<input type="checkbox"/> Apple TV			

Wir empfehlen hier **6** Geräte pro Testlauf. Das entspricht ca. **5** Testern für einen Tag.
Der Preis bewegt sich zwischen **2.480 - 3.360 Euro** (zzgl. einmalig **1.440 Euro** für die Testplanerstellung.)

Abbildung 5.1: Preisvorschlag für einen einmaligen Test einer mittelgroßen Webseite
(vgl. Appmatics | App-Testing und mobile Web-Testing, 2806)

Ein weiterer Vorteil ist der Kostenfaktor. Die Prüfung der Website durch externe Anbieter beinhaltet hohe, fortlaufende Kosten. So kostet beispielsweise eine einmalige Testung einer mittelgroßen Webseite durchschnittlich 2900 Euro zuzüglich einer Pauschale von rund 1400 Euro für die Erstellung eines Testplans. Bei einer monatlichen Testung durchschnittlich 2700 Euro pro Monat und bei einer quartalsweisen Testung durchschnittlich 2800 Euro pro Release. Dabei werden rund sechs Geräte und fünf Tester:innen benötigt (siehe Abbildung 5.1). Durch die interne Erstellung von Test können Kosten gering gehalten werden und sowohl schnell als auch einfach Änderungen vorgenommen und neue Tests erstellt werden.

Folgernd steigern die Tests also die Softwarequalität, erfüllen in hohem Maße die Qualitätsanforderungen der ISO 27001, sind günstiger als externe Tests für Webanwendungen und sorgen für eine erhöhte Zufriedenheit bei den Kund:innen. Deshalb sollten weiterhin Tests für Webanwendungen intern entwickelt werden, wenn der Umfang für das manuelle Testen zu hoch ist.

5.2 Aussicht

Dieser Abschnitt behandelt die Aussichten für automatisierte Tests von Webanwendungen im Rahmen des SIMK-Projekts.

5.2.1 Tests für weitere Institute

Da die Websites der unterschiedlichen Institute den gleichen Grundaufbau haben und sich nur in Dingen wie dem Namen, Bildern und Texten unterscheiden, können die Selenium Test universell für alle Institute eingesetzt werden. Hierbei zeigt sich nochmal deutlich der Vorteil der Runsettings Datei, da dort nur die URL und Zugangsdaten geändert werden müssen.

5.2.2 Tests für die neue Webseite

Das Grundgerüst der S-Erleben-Website wurde überarbeitet. Einige Institute wurden schon umgestellt und weitere sollen folgen. Da das neue Layout dem alten ähnelt, können einige Methoden übernommen werden und bei anderen müssen nur geringfügige Anpassungen vorgenommen werden.

Solange noch beide Layouts genutzt werden, werden die entwickelten Tests für das alte Layout weiterhin genutzt, erst wenn alle Institute umgestellt worden sind, sollen diese abgestellt und komplett durch die Tests für das neue Layout ersetzt werden.

5.2.3 Testumgebung

Zukünftig sollen die Tests nicht auf PROD getestet werden, sondern täglich in einer separaten Testumgebung, die der Integrationsumgebung entspricht. Somit wird auch das Pensum angehoben.

5.2.4 Tests für weitere Websites

Da in anderen Abteilungen von Oevermann auch noch weitere Webanwendungen implementiert werden, kann überlegt werden, für diese Websites auch automatisierte Tests zu entwickeln. Dies würde allgemein dazu führen, dass Fehler schneller und häufiger gefunden und behoben werden, was zu einer erhöhten Kundenzufriedenheit führen würde.

5.3 Themen für Praxisprojekte und Bachelorarbeiten

Themen für weitere Praxisprojekte und Bachelorarbeiten könnten sein:

- Vergleich von Testtools für Webanwendungen
- Vergleich von Testframeworks für Webanwendungen
- Erstellung von Tests für die neue Website
- Selenium IDE und Grid verwenden

Abbildungsverzeichnis

1.1	S-Erleben Webseite	3
1.2	S-Erleben Webseite eingeloggt	4
1.3	TÜV ISO 27001 Zertifizierungsprozess (TÜV-Rheinland)	7
2.1	Selenium: direkte Verbindung (vgl. Selenium, 0712b)	10
2.2	Selenium: remote Verbindung (vgl. Selenium, 0712b)	11
2.3	Selenium: remote Verbindung mit einem Testframework (vgl. Selenium, 0712b)	11
3.1	Verwendete NuGet Pakete	14
3.2	Prozessdiagramm des Kontaktformulars der S-Erleben-Website (1)	15
3.3	Prozessdiagramm des Kontaktformulars der S-Erleben-Website (2)	16
3.4	Neue Instanz eines ChromeDrivers	17
3.5	Kontaktseite der S-Erleben-Website	19
3.6	Kontaktformular der S-Erleben-Website	20
3.7	Dropdown des Kontaktformulars der S-Erleben-Website	23
3.8	Ausgefülltes Kontaktformular der S-Erleben-Website	25
3.9	Seite mit Bestätigungstext nach Absenden des Kontaktformulars	26
3.10	Test-Explorer mit den Testklassen und Testmethoden	36
3.11	Shop der S-Erleben-Website	40
4.1	Ein Release der SIMK Projektpipeline mit den unterschiedlichen Stages	44
4.2	Testübersicht einer Stage in Azure DevOps	45
4.3	Detaillierte Testresultate: Debug	45
4.4	Detaillierte Testresultate: Attachments	46
4.5	Clientfehler: 404	49
4.6	Serverfehler: 500	50
4.7	Fehlerhafte Darstellung eines Kalenders	50
5.1	Preisvorschlag für einen einmaligen Test einer mittelgroßen Webseite (vgl. Appmatics App-Testing und mobile Web-Testing, 2806)	52

Tabellenverzeichnis

2.1	Broser	11
3.1	IWebDriverMethoden	18
3.2	einfacheSelektoren	23
3.3	Testattribute	28

Literaturverzeichnis

[Sel] SELENIUM (Hrsg.): *WebDriverWait Class.* https://www.selenium.dev/selenium/docs/api/dotnet/html/T_OpenQA_Selenium_Support_UI.WebDriverWait.htm. – zuletzt geprüft am 03.01.2022

[Albahari u. Bahari 2018] ALBAHARI, Joseph ; ALBAHARI, Ben: *C# 7.0 - kurz & gut*. 5. Heidelberg : O'Reilly Verlag, 2018 (kurz & gut). <https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=5371904>. – ISBN 9783960101741

[Appmatics | App-Testing und mobile Web-Testing 2806] APPMATICS | APP-TESTING UND MOBILE WEB-TESTING: *Appmatics Preiskalkulator für App-Testing und mobile Web-Testing*. <https://www.appmatics.de/preiskalkulator/>. Version: 28.06.2021. – zuletzt geprüft am 17.01.2022

[Chromium 1904] CHROMIUM: *ChromeDriver - WebDriver for Chrome - Getting started*. <https://chromedriver.chromium.org/getting-started>. Version: 19.04.2021. – zuletzt geprüft am 31.12.2021

[Chromium 3012] CHROMIUM: *ChromeDriver - WebDriver for Chrome*. <https://chromedriver.chromium.org/>. Version: 30.12.2021. – zuletzt geprüft am 30.12.2021

[DIN EN ISO/IEC 27001 2017] DIN EN ISO/IEC 27001: *Informationstechnik - Sicherheitsverfahren - Informationssicherheitsmanagementsysteme - Anforderungen (ISO/IEC 27001:2013 einschließlich Cor 1:2014 und Cor 2:2015); Deutsche Fassung EN ISO/IEC 27001:2017*. <https://www.perinorm.com/document.aspx>. Version: 2017-06. – zuletzt geprüft am 29.06.2021

[DSV Gruppe 3011] DSV GRUPPE: *S Direkt und MehrWert Servicegesellschaft sind jetzt „S-Markt & Mehrwert“*. <https://ueberuns.dsv-gruppe.de/presse/presseinformationen/archiv/20181130-s-direkt-mwgs-fusionieren-s-markt-und-mehrwert/index.htm>. Version: 30.11.2018. – zuletzt geprüft am 29.06.2021

[Hoffmann 2013] HOFFMANN, Dirk W.: *Software-Qualität*. 2., aktualisierte u. korr. Aufl. 2013. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013 (eXamen.press). <http://dx.doi.org/10.1007/978-3-642-35700-8>. – ISBN 978-3-642-35700-8

[Hoffmann u. Lienhart 2008] HOFFMANN, Simon ; LIENHART, Rainer: *OpenMP*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. <http://dx.doi.org/10.1007/978-3-540-73123-8>. – ISBN 978-3-540-73122-1

Literaturverzeichnis

- [Jonathan Robie 2212] JONATHAN ROBIE ; W3 (Hrsg.): *What is the Document Object Model?* <https://www.w3.org/TR/WD-DOM/introduction.html>. Version: 22.12.2014. – zuletzt geprüft am 01.01.2022
- [Microsoft] MICROSOFT: *Assert Class (Microsoft.VisualStudio.TestTools.UnitTesting)*. <https://docs.microsoft.com/de-de/dotnet/api/microsoft.visualstudio.testtools.unittesting.assert?view=visualstudiosdk-2022>. – zuletzt geprüft am 02.01.2022
- [Microsoft 0212] MICROSOFT: *Was ist NuGet, und welche Funktion hat es?* <https://docs.microsoft.com/de-de/nuget/what-is-nuget>. Version: 02.12.2021. – zuletzt geprüft am 31.12.2021
- [Microsoft 1412] MICROSOFT: *Konfigurieren von Komponententests mit einer RUNSETTINGS-Datei.* <https://docs.microsoft.com/de-de/visualstudio/test/configure-unit-tests-by-using-a-dot-runsettings-file?view=vs-2022>. Version: 14.12.2021. – zuletzt geprüft am 02.01.2022
- [Microsoft 1611] MICROSOFT: *Anatomy of a Unit Test.* [https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms182517\(v=vs.90\)](https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/ms182517(v=vs.90)). Version: 16.11.2012. – zuletzt geprüft am 02.01.2022
- [Microsoft 2007] MICROSOFT: *Guidelines for Smoke Testing.* [https://docs.microsoft.com/en-gb/previous-versions/ms182613\(v=vs.80\)?redirectedfrom=MSDN](https://docs.microsoft.com/en-gb/previous-versions/ms182613(v=vs.80)?redirectedfrom=MSDN). Version: 2007. – zuletzt geprüft am 29.06.2021
- [Microsoft 2021] MICROSOFT: Microsoft Lösungsanbieter: OEVERMANN Networks GmbH. https://www.microsoft.com/de-de/solution-providers/partnerdetails/oevermann-networks-gmbh_bfee5054-aaf9-43e3-90b1-800d395dcf4e/da2e18f2-ec6a-4483-ba6d-7259aa8dd9e3. Version: 2021. – zuletzt geprüft am 22.06.2021
- [Microsoft 2709] MICROSOFT: *Verwenden von Objekten, die IDisposable implementieren.* <https://docs.microsoft.com/de-de/dotnet/standard/garbage-collection/using-objects>. Version: 27.09.2021. – zuletzt geprüft am 05.01.2022
- [Microsoft 2809] MICROSOFT: *try-finally – C#-Referenz.* <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/try-finally>. Version: 28.09.2021. – zuletzt geprüft am 02.01.2022
- [.NET Blog 2019] .NET BLOG: *Announcing the .NET Framework 4.8.* <https://devblogs.microsoft.com/dotnet/announcing-the-net-framework-4-8/>. Version: 2019. – zuletzt geprüft am 22.06.2021
- [OEVERMANN Networks GmbH 2021a] OEVERMANN NETWORKS GMBH: *Bankensoftware: Software für Finanzwirtschaft / Banken / OEVERMANN.* <https://www.oevermann.de/finanz-services.aspx>. Version: 2021. – zuletzt geprüft am 22.06.2021

Literaturverzeichnis

[OEVERMANN Networks GmbH 2021b] OEVERMANN NETWORKS GMBH: *Top-Referenz von OEVERMANN: Kreissparkasse Köln.* <https://www.oevermann.de/kreissparkasse-koeln-referenz.aspx>. Version: 2021. – zuletzt geprüft am 22.06.2021

[Osheroove u. a. 2015] OSHEROVE, Roy ; FEATHERS, Michael ; MARTIN, Robert C.: *The art of unit testing.* 2nd ed. Shelter Island, NY : Manning Publications, 2015 https://content-select.com/media/moz_viewer/5c85864b-99ac-40a2-8920-6037b0dd2d03/language:de. – ISBN 9783826687211. – zuletzt geprüft am 14.01.2022

[S-Markt & Mehrwert GmbH & Co. KG 1812] S-MARKT & MEHRWERT GMBH & Co. KG: *Sparkassen-Finanzgruppe bündelt Dienstleister für Markt- und Mehrwertservices: S-IMK und S-Markt & Mehrwert gehen zusammen.* https://www.sparkassedirekt.de/fileadmin/s-direkt/dateien/Medieninformation_Zusammenschluss_S-Markt_und_Mehrwert_mit_S-IMK.pdf. Version: 18.12.2018. – zuletzt geprüft am 28.06.2021

[S-Markt & Mehrwert GmbH & Co. KG 2019] S-MARKT & MEHRWERT GMBH & Co. KG: *Allgemeine Teilnahmebedingungen für das Kundenportal „S-Erleben“.* <https://www.erleben-ist-einfach.de/ksk/allgemeine-teilnahmebedingungen.pdfx?forced=true&forced=true>. Version: 2019. – zuletzt geprüft am 26.08.2019

[Selenium] SELENIUM: *Package org.openqa.selenium.* <https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/package-summary.html>. – zuletzt geprüft am 01.01.2022

[Selenium 0401] SELENIUM: *Ecosystem.* <https://www.selenium.dev/ecosystem/>. Version: 04.01.2022. – zuletzt geprüft am 05.01.2022

[Selenium 0712a] SELENIUM: *Avoid sharing state.* https://www.selenium.dev/documentation/test_practices/encouraged/avoid_sharing_state/. Version: 07.12.2021. – zuletzt geprüft am 03.01.2022

[Selenium 0712b] SELENIUM: *Selenium components.* <https://www.selenium.dev/documentation/overview/components/>. Version: 07.12.2021. – zuletzt geprüft am 06.01.2022

[Selenium 0712c] SELENIUM: *Shared capabilities.* <https://www.selenium.dev/documentation/webdriver/capabilities/shared/#eager>. Version: 07.12.2021. – zuletzt geprüft am 03.01.2022

[Selenium 0712d] SELENIUM: *Waits.* <https://www.selenium.dev/documentation/webdriver/waits/>. Version: 07.12.2021. – zuletzt geprüft am 03.01.2022

[Selenium 0812] SELENIUM: *Working with windows and tabs.* <https://www.selenium.dev/documentation/webdriver/browser/windows/>. Version: 08.12.2021. – zuletzt geprüft am 02.01.2022

Literaturverzeichnis

- [Selenium 1212] SELENIUM: *Install browser drivers.* https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/. Version: 12.12.2021. – zuletzt geprüft am 31.12.2021
- [Selenium 1312] SELENIUM: *Tips on working with locators.* https://www.selenium.dev/documentation/test_practices/encouraged/locators/. Version: 13.12.2021. – zuletzt geprüft am 05.01.2022
- [Selenium 2021a] SELENIUM: *Announcing Selenium 4.* <https://www.selenium.dev/blog/2021/announcing-selenium-4/>. Version: 2021. – zuletzt geprüft am 30.12.2021
- [Selenium 2021b] SELENIUM: *Getting started.* https://www.selenium.dev/documentation/webdriver/getting_started/. Version: 2021. – zuletzt geprüft am 30.12.2021
- [Selenium 2021c] SELENIUM: *Selenium Grid 4.* <https://www.selenium.dev/documentation/grid/>. Version: 2021. – zuletzt geprüft am 30.12.2021
- [Selenium 2021d] SELENIUM: *Selenium overview.* <https://www.selenium.dev/documentation/overview/>. Version: 2021. – zuletzt geprüft am 29.12.2021
- [Selenium 2112] SELENIUM: *Mouse Actions.* https://www.selenium.dev/documentation/webdriver/actions_api/mouse/. Version: 21.12.2021. – zuletzt geprüft am 01.01.2022
- [Selenium 2412a] SELENIUM: *By.* <https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/By.html>. Version: 24.12.2021. – zuletzt geprüft am 01.01.2022
- [Selenium 2412b] SELENIUM: *Package org.openqa.selenium.chrome.* <https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/chrome/package-summary.html>. Version: 24.12.2021. – zuletzt geprüft am 01.01.2022
- [Selenium IDE 1205] SELENIUM IDE: *Command-line Runner.* <https://www.selenium.dev/selenium-ide/docs/en/introduction/command-line-runner>. Version: 12.05.2021. – zuletzt geprüft am 05.01.2022
- [Tim Bray u. a. 2008] TIM BRAY ; JEAN PAOLI ; C. M. SPERBERG-MCQUEEN ; EVE MALER, Sun M. ; FRANÇOIS YERGEAU ; W3C (Hrsg.): *Extensible Markup Language (XML) 1.0 (Fifth Edition).* <https://www.w3.org/TR/xml/>. Version: 2008. – zuletzt geprüft am 03.01.2022
- [TÜV-Rheinland] TÜV-RHEINLAND: *ISO 27001 Zertifizierung – Informationssicherheit.* <https://www.tuv.com/germany/de/informationssicherheit-iso-27001.html>. – zuletzt geprüft am 22.06.2021
- [W3 Schools 0101] W3 SCHOOLS ; REFSNES DATA (Hrsg.): *CSS Selectors.* https://www.w3schools.com/css/css_selectors.asp. Version: 01.01.2022. – zuletzt geprüft am 01.01.2022

Literaturverzeichnis

- [Zhan 2015] ZHAN, Zhimin: *Selenium WebDriver Recipes in C#*. Second Edition. Berkeley, CA : Apress, Imprint: Apress, 2015. <http://dx.doi.org/10.1007/978-1-4842-1742-9>. – ISBN 978-1-4842-1742-9