



Универзитет „Св. Кирил и Методиј“ во Скопје  
**ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И  
КОМПЈУТЕРСКО ИНЖЕНЕРСТВО**

Документација за проектот по предметот  
Софтверски квалитет и тестирање  
на тема:

## **Тестирање на Booktopia – .NET + React апликација за управување со книги и автори**

*Изработено од: Тамара Стојанова*

---

Датум на изработка:  
22.09.2024

Линк до Git репозиториумот со source кодот и тестовите:  
<https://github.com/tamarastojanova/booktopia-sqat>

## Содржина

Содржина .....	2
Вовед .....	3
Технологии користени при изработка на апликацијата.....	3
Алатки и технологии користени при тестирање на апликацијата.....	5
1. End-to-end тестови .....	5
2. Integration тестови.....	6
3. Unit тестови .....	6
→ Валидациски extension методи.....	7
→ Endpoint-и на REST контролерите.....	8
→ Методи во Repository слојот .....	10
Заклучок.....	11

## Вовед

Сеопфатното тестирање на софтверските апликации е од суштинска важност за обезбедување на квалитетот, стабилноста и безбедноста на софтверските производи. Овој процес вклучува систематско проверување на сите аспекти на апликацијата, од функционалностите до перформансите и безбедносните аспекти. Тестирањето помага при осигурување на квалитетот на крајниот производ и неговата усогласеност со релевантните индустриски стандарди и регулативи, го зголемува задоволството на корисниците и нивната доверба во производот, и, можеби најважното за софтверските компании ширум светот – ги намалува трошоците заради откривањето и поправката на грешките во порани фази од развојот на софтверот.

Ваквото правилно тестирање е наменето и за помали и за поголеми апликации, бидејќи во секоја апликација, без разлика на нејзината сложеност, може да се јават критични недостатоци. Всушност, темата на овој проект е тестирање на прилично едноставна CRUD апликација за менаџирање со автори и книги, што го доловува значењето на тестирањето дури и за основни софтверски решенија. Изработката на овој проект уште повеќе ми ја разјасни потребата од софтверското тестирање, бидејќи иако на почетокот навидум сè функционираше како што треба, на некои сегменти им беше неопходно рефакторирање и подобрување.

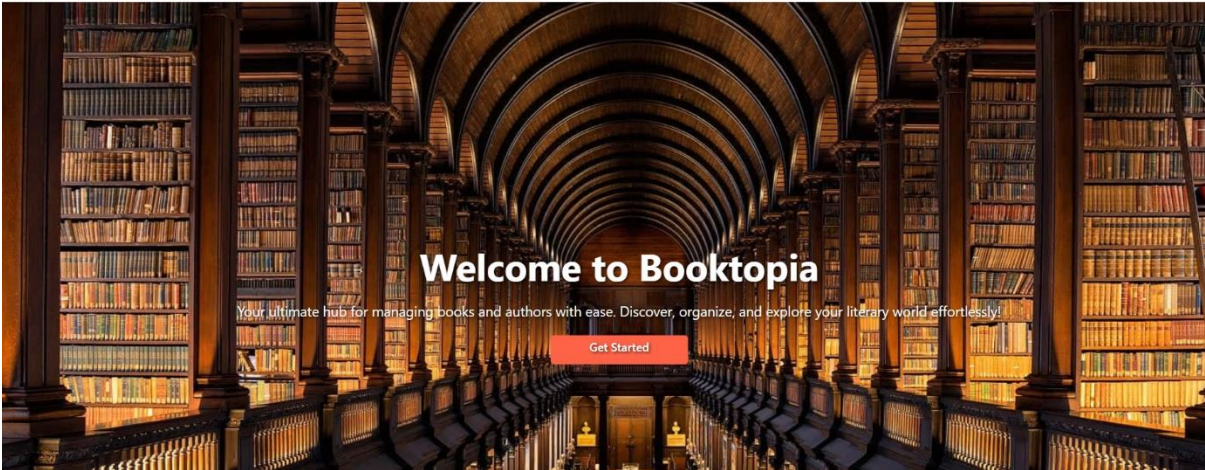
## Технологии користени при изработка на апликацијата

Апликацијата која е предмет на тестирање, Booktopia, се состои од backend и frontend проект, и е изработена за време на bootcamp во кој учествував. Backend делот е моја изработка, додека frontend делот е изработен од колега. Backend-от, кој е напишан во .NET C#, е всушност веб апликација со Swagger UI кој овозможува праќање на барања и добивање на одговори од REST контролери за автори и книги. Оваа апликација е организирана во слоевита т.н. Onion архитектура составена од веб слој, сервисен слој, слој кој симулира конекција со база на податоци, и слој кој се однесува на доменот на апликацијата. Апликацијата не е поврзана со вистинска база на податоци, па затоа третиот слој ова го заменува со статичка in-memory листа од ентитети. Frontend проектот е изработен во React со помош на Vite. Апликацијата не поддржува автентикација и авторизација.

Во прилог ќе оставам неколку фотографии од корисничкиот интерфејс на Booktopia.

Booktopia

HomeAuthors



# Welcome to Booktopia

Your ultimate hub for managing books and authors with ease. Discover, organize, and explore your literary world effortlessly!

Get Started


Booktopia

HomeAuthors

Search


Authors

Add Author




William Shakespeare

ViewEditDelete




Agatha Christie

ViewEditDelete



Barbara Cartland

ViewEditDelete



Harold Robbins

ViewEditDelete

Activate Windows  
Go to Settings to activate Windows.

Booktopia

HomeAuthors

Add Book



William Shakespeare

Books

Romeo and Juliet

Year published: 1597

EditDelete

Hamlet

Year published: 1600

EditDelete

Othello

Year published: 1603

EditDelete

Activate Windows  
Go to Settings to activate Windows.

## Алатки и технологии користени при тестирање на апликацијата

За темелно тестирање на апликацијата, имплементирани се повеќе различни видови на тестови:

### 1. End-to-end тестови

End-to-end (E2E) тестирањето е метод за тестирање на софтвер кој ги проверува функционалностите и интеграцијата на целата апликација, од почеток до крај. Целта на E2E тестирањето е да се осигура дека сите делови на системот работат заедно онака како што е очекувано, и дека апликацијата ги задоволува сите барања од перспектива на корисникот. Тоа обично опфаќа тестирање на корисничкиот интерфејс, базата на податоци, серверските компоненти и сите меѓусебни интеракции. E2E тестирањето обично се имплементира во доцните фази од развојниот процес, кога апликацијата е релативно стабилна и сите нејзини делови се интегрирани.

За E2E тестирање на апликацијата користев Cypress. Cypress е современа алатка за тестирање на frontend базирана на JavaScript која е специјално дизајнирана за да ги олесни и автоматизира тестовите за веб апликациите. Оваа алатка поддржува модерни технологии за развој на веб апликации, како React, Angular, Vue.js итн. Со Cypress лесно се пишуваат и извршуваат тестови кои симулираат кориснички интеракции со апликацијата, како кликување на копчиња, пополнување на форми или навигација низ различни страници.

Го користев Cypress за тестирање на различни сценарија во апликацијата, како тестирање на навигацијата меѓу страниците, извршување на CRUD операциите според можностите што ги нуди корисничкиот интерфејс, како и проверка на точноста на податоците што се прикажуваат на истиот по извршувањето на ваквите операции. Cypress е добар за оваа намена бидејќи нуди едноставно и интуитивно API, вградено зачувување на тестови и одлична поддршка за дебагирање. Cypress се извршува директно во прелистувачот, што го прави брз и сигурен за тестирање на динамични веб апликации.

Конфигурирањето на Cypress беше релативно едноставно – ја инсталирав алатката и ја стартував, при што се отвори Cypress Test Runner-от каде можеме да креираме и извршуваме тестови. По поставување на одредени конфигурациски параметри, како URL-от на којшто може да се пристапи до React апликацијата, околината за извршување на тестовите беше сетирана.

Cypress всушност се покажа одличен за тестирање на React frontend-от, бидејќи ми укажа на одредени недостатоци кои требаше да бидат поправени. При симулација на некои функционалности, во output-от на неколку тестови приметив дека се прават бесконечен број на повици до одредена патека, што многу ги влошуваше перформансите и го закочуваше и самиот Test Runner. Ова ме доведе до заклучок дека е формирана јамка од повици заради неправилно менаџирање со состојбите во React, што би било тешко да се забележи при мануелно тестирање на апликацијата. Воедно,

кодот потребен за симулација на корисник кој е во интеракција со апликацијата е многу едноставен и разбирлив.

## 2. Integration тестови

Интеграциските тестови се тестови кои се фокусираат на проверка на комуникацијата и интеракцијата меѓу различни компоненти на системот. Иако тесно поврзани и често поистовестувани со E2E тестовите, за разлика од нив, кои ја тестираат целата функционалност на апликацијата „од крај до крај“, интеграциските тестови се концентрираат на тестирање на одредени делови или модули на системот, осигурувајќи се дека тие работат заедно како што е предвидено. Интеграциските тестови обично се изведуваат во средните фази од развојот на производот, кога различни модули се интегрираат, но пред да се извршат целосните E2E тестови. Тие овозможуваат рано откривање на проблеми со интеграција, па истите може да се решат пред да станат критични.

За потребите на проектот имплементирани се интеграциски тестови во XUnit на контролерите со цел да го провериме праќањето на барања до нив преку користење на таканаречен клиент и CustomWebFactory. CustomWebFactory се користи за креирање и конфигурирање на специјализирана средина за тестирање, што ни овозможува симулирање на барања до API контролерите без потреба од вистински сервер. На овој начин, можеме да провериме дали контролерите правилно ги обработуваат барањата и враќаат соодветни резултати.

Дополнително, направени се и интеграциски тестови на сервисите за проверка на нивната интеграција со repository слојот. Овие тестови ни овозможуваат да се осигураме дека сервисите правилно го користат соодветното repository за читање и запишување на податоци. За да се постигне независност и изолација меѓу секој test case, имплементирав метод за ресетирање во соодветното repository, кој ја враќа статичката листа од податоци во првобитна состојба. Ова овозможува тестовите да даваат конзистентни резултати и по нивно повеќекратно извршување. Покрај овој метод, за потребите на понатамошните тестови имплементиран е и метод за сетирање на оваа статичка листа, со цел поробустен код и овозможување на поголема контрола врз податоците при креирање на соодветниот тест. Всушност, интеграциските тестови направени врз сервисниот слој можат да се сметаат и како unit тестови за истиот, бидејќи во него нема никаква дополнителна логика освен пропагирање на барањето до repository слојот, кој ја извршува манипулацијата со податоците.

Во овие интеграциски тестови не користев специфичен тип на coverage бидејќи истите ги разбираам како тестови кои ја проверуваат комуникацијата помеѓу неколку компоненти од апликацијата, а не деталната логика на методите. Затоа, покривањето на логиката со одреден тип на coverage е вклучено во наредната група на тестови.

## 3. Unit тестови

Unit тестирањето е процес каде што се тестира најмалата функционална целина на код. Ваквите тестови се неопходни за верификација на функционалноста на индивидуалните компоненти или методи во рамките на кодот. Тие нè осигуруваат дека секој единичен дел од кодот работи правилно во изолација, овозможувајќи им

на developerите да идентификуваат и корегираат проблеми и грешки уште во најраните фази од развојот. Понентата на ваквите тестови најчесто не е само да тестираат дека логиката го прави она што е потребно да го прави, туку и да се осигураме дека со промените кои сме ги направиле во кодот не сме скршиле постоечка функционалност на системот. Со редовно извршување на unit тестовите, developerите можат безбедно да го рефакторираат кодот, да додаваат нови функционалности и да прават подобрувања без страв од воведување нови грешки. Unit тестовите за целините во овој проект се креирани во NUnit.

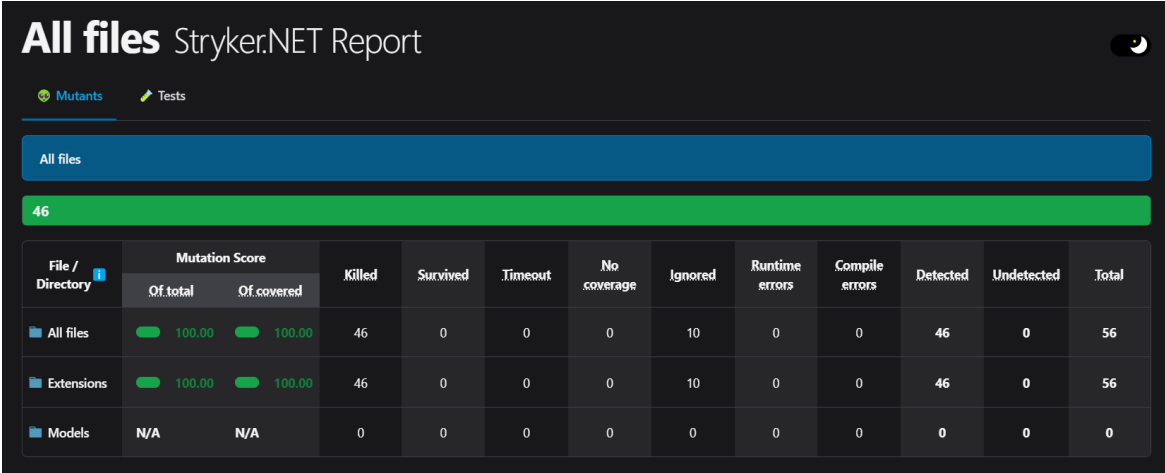
## → Валидациски extension методи

Најпрво, креирам unit тестови за двете валидациски extension методи за ентитетите, кои вклучуваат повеќе условно рендерирање. За квалитетно да се истестираат овие методи, користев со покривање базирано на синтакса, односно со *креирање на мутанти* и генерирање на тестови кои истите ќе ги убијат. За оваа цел ја користев Stryker алатката за .NET.

Stryker е алатка за мутантно тестирање за .NET која помага во обезбедувањето на квалитетот на unit тестовите. Ваквиот тип на тестирање вклучува мали модификации на кодот (креирање мутанти) и потоа извршување на тестовите за да се види дали тие ќе ги откријат промените. Ако мутантот преживее, тоа укажува на потенцијална слабост во тестовите.

За конфигурирање на Stryker, може да се наведе кои методи или класи треба да се мутираат и каде се лоцирани соодветните тестови кои ќе се обидат да ги убијат генерираните мутанти. Stryker потоа одлучува кои мутанти ќе ги дефинира за одреден метод преку правење мали промени, како што се менување на оператори или слична модификација на контролниот тек. Целта е да се креира најмало можно множество од тестови кои ќе ги убијат сите мутанти, осигурувајќи темелно покривање на кодот.

Разгледувајќи ги новокреираните мутанти, успеав да дефинирам тестови кои ќе ги „покријат“ ваквите промени и ќе докажат дека постојат примери кога ваквата ситна промена во кодот доведува до излез различен од очекуваниот. На сликата во прилог може да се согледа бројот на убиени мутанти, проследен со високиот Mutation score.



**All files Stryker.NET Report**

46

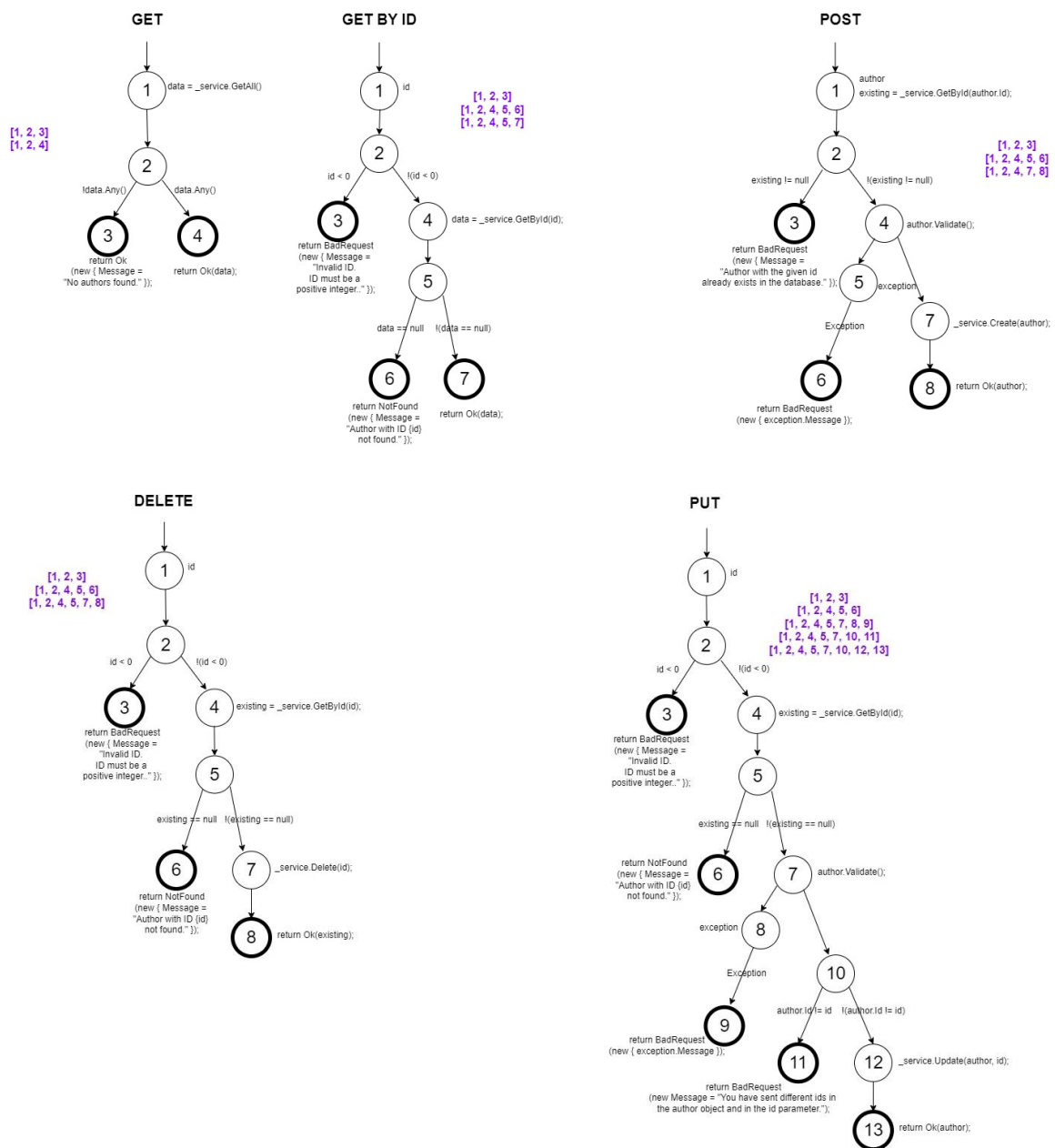
File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	100.00	100.00	46	0	0	0	10	0	0	46	0	56
Extensions	100.00	100.00	46	0	0	0	10	0	0	46	0	56
Models	N/A	N/A	0	0	0	0	0	0	0	0	0	0



## → Endpoint-и на REST контролерите

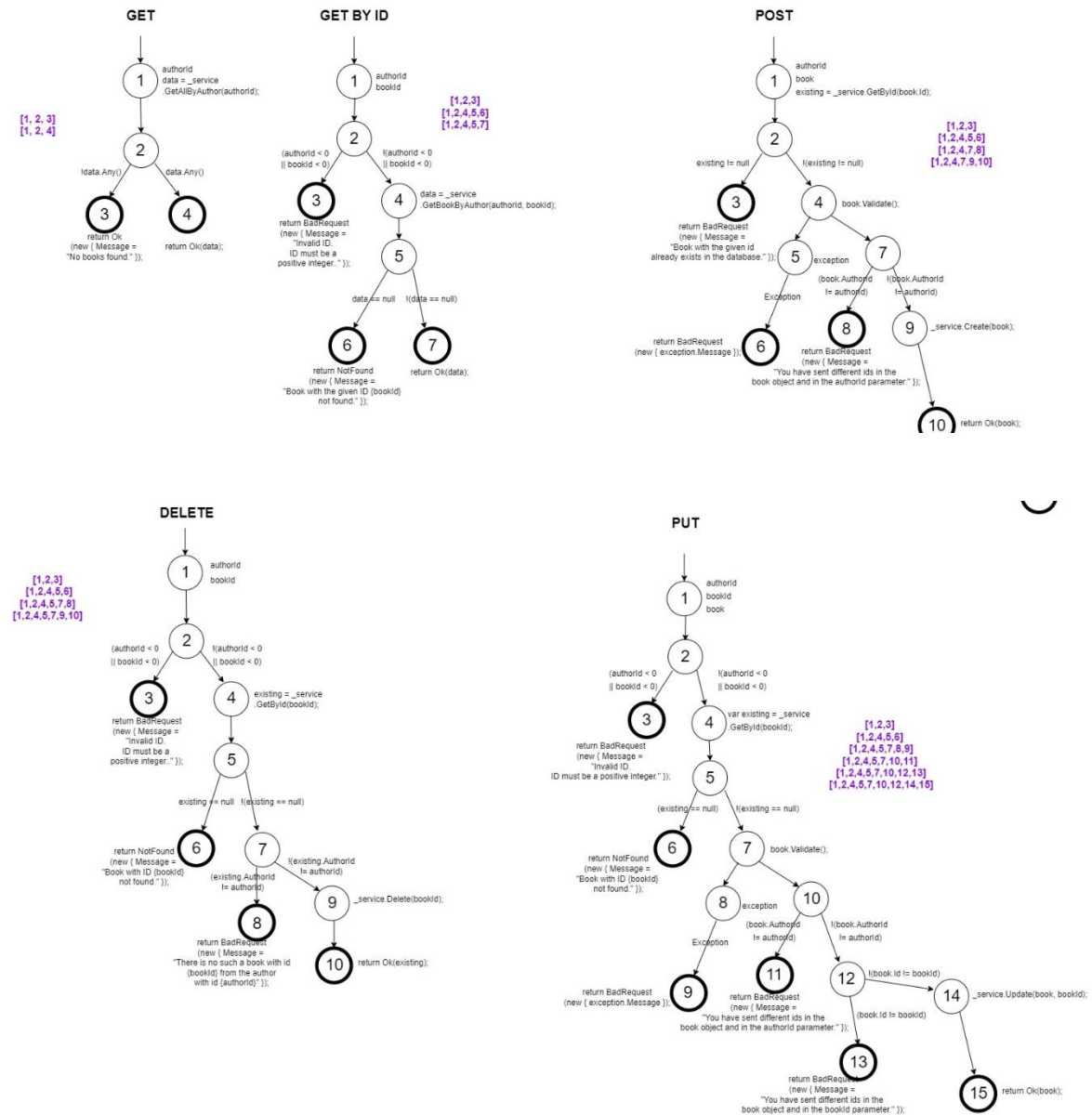
Следен чекор беше креирање на unit тестови за сите методи од кои се составени API контролерите за автори и книги, кои исто така вклучуваа условно рендерирање. За покривање на целата логика кои овие методи ја имплементираат, се одлучив да применам покривање базирано на графови кои го опишуваат контролниот тек на програмата, но со фокус на различни специфични критериуми.

За тестирање на AuthorsController-от избрав *Prime Path Coverage*, што овозможува детекција на основните патишта во кодот и помага во обезбедување на покриеност на клучните логички структури. На следните фотографии се прикажани граф моделите кои ги направив за оваа намена, а означени со виолетова боја се тест патеките користени при кодирањето на соодветните unit тестови.





Во однос на BooksController-от, се одлучив за *Data Flow Coverage*, со акцент на All def-use paths, што овозможува следење и анализирање на променливите во текот на извршување на кодот, осигурувајќи се дека сите патишта што ја дефинираат и користат променливата се покриени. На следните фотографии, пак, може да се забележат моделите креирани за овие тестови.



Виолетовите патеки се соодветно дефинираните тест патеки, а постапката по која истите се добиени ќе ја оставам во прилог, но само за најкомплексниот метод од горенаведените – PUT.

<pre> * defs: authorId 1 bookId 1 book 1 existing 4 exception 8  * uses: authorId 2,3 2,4 10,11 10,12 bookId 2,3 2,4 4 6 12,13 12,14 14 book 7 10,11 10,12 12,13 12,14 14 15 existing 5,6 5,7 exception 9 </pre>	<pre> * DU Pairs: authorId [1, (2, 3)], [1, (2, 4)], [1, (10, 11)], [1, (10, 12)] bookId [1,4], [1,6], [1,14], [1, (2, 3)], [1, (2, 4)], [1, (12, 13)], [1, (12, 14)] book [1,7], [1,14], [1,15], [1, (10, 11)], [1, (10, 12)], [1, (12, 13)], [1, (12, 14)] existing [4, (5, 6)], [4, (5, 7)] exception [8,9]  * DU Paths: authorId [1,2,4], [1,2,3], [1,2,4,5,7,10,12], [1,2,4,5,7,10,11] bookId [1,2,4], [1,2,3], [1,2,4,5,6], [1,2,4,5,7,10,12,14], [1,2,4,5,7,10,12,13] book [1,2,4,5,7], [1,2,4,5,7,10,12], [1,2,4,5,7,10,11], [1,2,4,5,7,10,12,14], [1,2,4,5,7,10,12,13], [1,2,4,5,7,10,12,14,15] existing [4,5,7], [4,5,6] exception [8,9] </pre>
--	---

- **ALL DU PATHS** (без дупликати, т.е. *тест патеки*): [1,2,3], [1,2,4,5,6], [1,2,4,5,7,8,9], [1,2,4,5,7,10,11], [1,2,4,5,7,10,12,13], [1,2,4,5,7,10,12,14,15].

Оваа комбинација на техники за покривање ми помогна да добијам поголема сигурност во квалитетот на тестирањето. На овој начин, успеав да креирам 16 различни тест случаи за методите од AuthorsController-от, и 19 различни тест случаи за методите од BooksController-от (по еден тест случај за секоја добиена тест патека).

## → Методи во Repository слојот

Последните неистестирани методи се наоѓаат во имплементациските класи во рамки на Repository слојот од Onion архитектурата. Логиката во овој дел од апликацијата е прилично едноставна, но многу важна за нејзиното очекувано функционирање, бидејќи истата се однесува на манипулацијата со статичката листа од податоци која во рамките на овој проект симулира база. Бидејќи тестирањето на кодот со мутантите генерирани од Stryker ми беше посебно интересно, одлучив и овие методи да ги истестирам на сличниот начин. Успешноста на креираните тестови да ги убијат мутантите може да се забележи на следната слика.

The screenshot shows the Stryker.NET Report interface. At the top, it says 'All files Stryker.NET Report'. Below that, there are tabs for 'Mutants' and 'Tests'. A blue bar indicates 'All files' and a green bar shows '48' mutants. The main table displays the following data:

File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	100.00	100.00	48	0	0	0	23	0	2	48	0	73
Implementation	100.00	100.00	48	0	0	0	23	0	2	48	0	73
Interface	N/A	N/A	0	0	0	0	0	0	0	0	0	0

## Заклучок

Во овој проект, фокусот беше ставен на генерирање на различни видови на тестови кои ќе бидат доволни за да нè осигураат дека софтверот којшто го развиваме ги има очекуваните функционалности и дека квалитетот на кодот е на високо ниво. Работењето со досега непознати технологии и алатки, често користени на проекти во многу од компаниите, беше интересен предизвик кој придонесе кон збогатување на постоечките знаења. Исто така, при изработката на овој проект успеав да стекнам поголемо искуство во оптимизирање на код, овозможување негова реискористливост, како и правилно именување на тестови со цел тие да бидат разбирливи за останатите, а притоа се потсетив и на изучуваните начини за покривање на кодот, кои навистина доведоа до негово темелно тестирање. Овој проект демонстрираше колку само неколку условни рендерирања можат да придонесат во комплексноста на кодот и неговата нечитливост.