

**Universiteti për Biznes dhe Teknologji**

**Master Shkenca Kompjuterike dhe Inxhinieri**



**Sistemi për Menaxhimin e Telefonave - SMT**

**Lënda: Inxhinieria e Softuerit për Aplikacione të Shkallëzueshme**

Profesoreshë:

Dr. Sc. Ermira Daka

Punoi:

Elma Redzepi

Prishtinë, 2025

# Sistemi për Menaxhimin e Telefonave - SMT

Elma Redzepi – [er242575418@ubt-uni.net](mailto:er242575418@ubt-uni.net)

University for Business and Technology, 10000 Prishtine, Kosovo

**Abstrakti:** Ky projekt synon të ofrojë një zgjidhje moderne dhe funksionale për menaxhimin e telefonave në një mjedis organizativ apo biznesor përmes zhvillimit të një sistemi të quajtur **SMT – Sistemi për Menaxhimin e Telefonave**. SMT është një aplikacion i ndërtuar mbi arkitekturën **Client–Server**, ku pjesa e backend-it është zhvilluar me **Slim PHP Framework**, ndërsa të dhënat ruhen në një **bazë të dhënash MySQL**. Pjesa frontend ndërvepron përmes API-ve RESTful të testuara me **Postman**, duke garantuar komunikim të sigurt dhe të qëndrueshëm midis përdoruesit dhe sistemit. Ky sistem mbështet të gjitha operacionet bazë **CRUD (Create, Read, Update, Delete)** për të menaxhuar të dhënat e telefonave si emri, marka, modeli, çmimi, sasia dhe gjendja. Struktura e bazës së të dhënave është ndërtuar në mënyrë të normalizuar për të mundësuar qasje efikase, integritet të të dhënave dhe zgjerueshmëri në të ardhmen. Për më tepër, përmes përdorimit të **phpMyAdmin** dhe **XAMPP**, sigurohet një ambient zhvillimi lokal stabil dhe i lehtë për menaxhim. Në projekt janë realizuar testime funksionale për të gjitha operacionet kryesore përmes Postman, duke konfirmuar saktësinë dhe integritetin e shërbimeve të ofruara nga API-të. Megjithatë testimet për shkallëzueshmëri (scalability) nuk janë përfshirë në mënyrë të plotë, struktura modulare e sistemit e bën atë të përgatitur për integritet të ardhshme dhe përpunim të sasive më të mëdha të të dhënave. SMT paraqet një shembull praktik të një aplikacioni të plotë me arkitekturë të ndarë dhe logjikë të qartë biznesore, i cili mund të shërbejë si model për sisteme të tjera të menaxhimit të produkteve.

# PËRMBJATJA

LISTA FIGURAVE .....	5
1. HYRJE .....	7
2. DIZAJNI ARKITEKTURËS – (HIGH-LEVEL ARCHITECTURE) .....	8
3. CI/CD Pipeline (Git Actions).....	10
4. DIZAJNI DATABAZES.....	11
5. IMPLEMENTIMI I APLIKACIONIT PËRMES POSTMAN-API BACK-end DHE FRONT-end..	12
<b>5.1. RESTful API – Backend.....</b>	<b>12</b>
<b>5.1.1. Paraqitja e të gjitha të dhënave .....</b>	<b>14</b>
<b>5.1.2. Paraqitja e një të dhëne të caktuar sipas ID-së .....</b>	<b>15</b>
<b>5.1.3. Shtimi i të dhënave .....</b>	<b>15</b>
<b>5.1.4. Modifikimi i të dhënave .....</b>	<b>17</b>
<b>5.1.5. Fshirja e të dhënave .....</b>	<b>18</b>
<b>5.2. RESTful API – Frontend.....</b>	<b>19</b>
<b>5.2.1. Paraqitja e të gjitha të dhënave .....</b>	<b>20</b>
<b>5.2.2. Kërkimi i të dhënave.....</b>	<b>21</b>
<b>5.2.3. Shtimi i të dhënave .....</b>	<b>22</b>
<b>5.2.4. Modifikimi i të dhënave .....</b>	<b>23</b>
<b>5.2.5. Fshirja e të dhënave .....</b>	<b>24</b>
6. CACHING AND PERFORMANCE OPTIMIZATION, MESSAGE QUEUES AND EVENT-DRIVEN ARCHITECTURE, FAULT TOLERANCE, HIGH AVAILABILITY, AND RESILIENCE...	26
<b>6.1. Caching and Performance Optimization .....</b>	<b>26</b>
<b>6.2. Message Queues and Event-Driven Architecture .....</b>	<b>26</b>
<b>6.3. Fault Tolerance .....</b>	<b>28</b>
<b>6.4. High Availability .....</b>	<b>28</b>
<b>6.5. Resilience .....</b>	<b>29</b>
7. SCALABILITY TESTS.....	30
<b>7.1. Test Scripts në Postman .....</b>	<b>30</b>
<b>7.2. Tests Scripts ne Postman per shtimin e testeve:.....</b>	<b>31</b>
<b>7.3. Tests Scripts ne Postman per modifikimin e testeve: .....</b>	<b>34</b>
<b>7.4. Tests Scripts ne Postman per modifikimin e testeve: .....</b>	<b>36</b>

8.	CLOUD INFRASTRUCTURE.....	39
9.	UNIT TESTS .....	44
10.	PËRFUNDIMI .....	45
	REFERENCAT.....	47

## LISTA FIGURAVE

Figure 1. Dizajni Arkitekturës .....	9
Figure 2. Databaza .....	11
Figure 3. Paraqitja e Postman – Backend .....	13
Figure 4. Paraqitja e të gjitha të dhënave – Backend .....	14
Figure 5. Paraqitja e një të dhënë – Backend .....	15
Figure 6. Shtimi i të dhënave – Backend .....	16
Figure 7. Modifikimi i të dhënave – Backend .....	18
Figure 8. Fshirja e të dhënave – Backend .....	19
Figure 9. Paraqitja e të dhënave – Frontend .....	20
Figure 10. Paraqitja e të dhënave – Frontend .....	21
Figure 11. Kërkimi i të dhënave – Frontend .....	22
Figure 12. Shtimi i të dhënave -Frontend .....	23
Figure 13. Modifikimi i të dhënave – Frontend .....	24
Figure 14. Fshirja e të dhënave – Frontend .....	25
Figure 15. Paraqitja e të dhënave me GET në Scripts .....	30
Figure 16. Shtimi i të dhënave POST .....	31
Figure 17. Gjenerimi i Scripts me POST .....	32
Figure 18. Scripts POST Run API .....	32
Figure 19. Gjenerimi i 100 Scripts – POST .....	33
Figure 20. Pas gjenerimit të 100 Scripts - Databaza .....	33
Figure 21. Paraqitja pas gjenerimit të Scripts edhe në Front-end .....	34
Figure 22. Tests Scripts - Modifikimi i një të dhëne .....	34
Figure 23. Gjenerimi i Scripts .....	35
Figure 24. Kemi gjeneruar 1 të dhënë me PUT .....	35
Figure 25. Gjenerimi i Scripts me PUT .....	36
Figure 26. Fshirja e në të dhëne .....	36
Figure 27. Gjenerimi i Scripts - DELETE .....	37
Figure 28. Gjenerimi i 10 Scripts me DELETE .....	37
Figure 29. Gjenerimi i të dhënave me DELETE .....	38
Figure 30. Performance Test Report .....	40

Figure 31. Response time.....	41
Figure 32. Throughput .....	42
Figure 33. Requests response times .....	42
Figure 34. Metrics .....	43
Figure 35. Error .....	43
Figure 36. Unit tests .....	45

## 1. HYRJE

Në epokën moderne të dixhitalizimit, menaxhimi efikas i pajisjeve teknologjike përbën një nevojë thelbësore për shumë biznese dhe institucione. Telefonat, si një prej mjeteve më të përdorura në komunikim dhe operacione të përditshme, kërkojnë një sistem të besueshëm për regjistrim, kontroll dhe përditësim të të dhënave. Për këtë arsye, u ndërtua projekti “**Sistemi për Menaxhimin e Telefonave – SMT**”, me qëllim të ofrojë një platformë të thjeshtë, por funksionale për trajtimin e informacionit në lidhje me telefonat.

SMT është një sistem me arkitekturë të ndarë, ku pjesa backend është ndërtuar duke përdorur **Slim PHP Framework**, i njohur për lehtësinë e tij dhe përkrahjen ndaj zhvillimit të API-ve RESTful. Këto API lejojnë komunikimin me pjesën frontend ose përdoruesin përmes kërkesave **GET, POST, PUT**, dhe **DELETE**. Si ambient zhvillimi është përdorur **XAMPP**, që integron serverin Apache dhe sistemin e menaxhimit të databazës MySQL për një përvojë testimi dhe zhvillimi sa më të mirë. Struktura e bazës së të dhënave përfshin tabela të dizajnuara për të mbështetur funksionalitetin e menaxhimit të telefonave, duke ruajtur fushat si: ***id, emri, marka, modeli, qmimi, sasia, dhe gjendja***. Në aspektin e testimeve, projekti është verifikuar funksionalisht përmes **Postman**, ku janë testuar të gjitha endpoint-et e API-së për të garantuar se çdo funksion i implementuar punon saktë. Për shembull, janë testuar shtimi i një telefoni të ri, përditësimi i të dhënave të një telefoni ekzistues, fshirja e telefonit, dhe marrja e listës së të gjithë telefonave. Këto testime demonstrojnë integrimin dhe qëndrueshmërinë e ndërveprimit mes komponentëve të sistemit. Qëllimi i këtij projekti nuk është vetëm të ofrojë një zgjidhje funksionale për menaxhimin e telefonave, por gjithashtu të demonstrojë përdorimin praktik të koncepteve të **Inxhinierisë së Softuerit**, përfshirë: analiza e kërkesave, dizajni i sistemit, implementimi i kodit, testimi dhe dokumentimi. SMT është ndërtuar në mënyrë të tillë që të jetë **lehtësisht i zgjerueshëm**, duke mundësuar në të ardhmen shtimin e funksionaliteteve si: kërkimi dinamik, kategorizimi i telefonave sipas markës, raportimi statistikor, apo autentifikimi i përdoruesve.

Ky projekt përfaqëson një hap të rëndësishëm drejt ndërtimit të sistemeve praktike që adresojnë nevoja konkrete dhe është i përshtatshëm për t'u përdorur si model për zgjidhje të tjera të ngjashme në fushën e menaxhimit të produkteve apo inventarit në organizata të ndryshme.

## 2. DIZAJNI ARKITEKTURËS – (HIGH-LEVEL ARCHITECTURE)

Dizajnimi i arkitekturës së këtij sistemi përfaqëson një qasje të strukturuar dhe të modularizuar, e cila e ndan sistemin në tri shtresa thelbësore:

- Ndërfaqja e përdoruesit (Front-End),
- Shtresa e logjikës së aplikacionit dhe shërbimeve (Back-End), si dhe
- Shtresa e ruajtjes së të dhënave (Database).

Kjo ndarje tre-shtresore është themelore për ndërtimin e aplikacioneve moderne, pasi garanton ndarje të përgjegjësi, mirëmbajtje më të thjeshtë, testim të pavarur të komponentëve dhe mundësi për zgjerim në të ardhmen pa ndikuar negativisht në pjesët tjera të sistemit.

Në pikënisje qëndron **përdoruesi**, i cili është subjekti aktiv i ndërveprimit me sistemin. Ai ndërvepron përmes ndërfaqes së ndërtuar me **Vue.js**, një framework progresiv për ndërtimin e ndërfaqeve të përdoruesit (SPA – Single Page Application), që siguron ngarkesë të shpejtë, përditësime dinamike të përmbajtjes, dhe përdorim të shtrirë të komponentëve të ripërdorshëm. Përmes formave, komandave dhe ndërfaqeve vizuale të Vue.js, përdoruesi nis kërkesa ndaj serverit në mënyrë asinkrone përmes **HTTP (AJAX/Fetch)**, duke ndarë qartë UI nga logjika serverike.

Kërkesat HTTP kapen nga një API e ndërtuar në **Slim PHP**, një mikro-framework që mbështet RESTful API, i cili është zgjedhur për lehtësinë, fleksibilitetin dhe performancën e tij. Slim vepron si ndërmjetës logjik dhe përpunues biznesi: i verifikon, i validon dhe i transformon të dhënat që vijnë nga Front-End-i dhe më pas merr vendime sipas rregullave të logjikës së brendshme të sistemit (si validimi i të dhënave të një pajisjeje, kontrolli i autentikimit, apo ndarja e të drejtave të aksesit për përdoruesit e ndryshëm). Slim gjithashtu orkestron komunikimin me bazën e të dhënave përmes pyetjeve **SQL**.

**Database Management System-i** i përdorur është **MySQL**, i cili është zgjedhur për stabilitetin, strukturën relacione dhe përputhjen me PHP. Të dhënat ruhen në mënyrë të normalizuar në tabela që përfaqësojnë entitete të ndryshme si: Telefonat\_SMT, Përdoruesit, Garancionet, SpecifikatTeknike etj. Struktura relacione garanton koherencë, integritet referencial dhe performancë të lartë në kërkesa të ndërlikuara me shumë bashkime (JOINS). Operacionet e CRUD-it (Create, Read, Update, Delete) mbështeten përmes një layer abstraksioni të ndërtuar mbi Slim PHP.

Lidhjet mes komponentëve përshkohen nga protokolle të standardizuara si **HTTP** për komunikim ndërmjet klientit dhe serverit, dhe **SQL** për ndërveprim me bazën e të dhënave. Për më tepër, sistemi është i integruar me mekanizma të automatizimit të zhvillimit si **CI/CD (Continuous Integration / Continuous Deployment)**. Përmes këtyre mekanizmave, çdo ndryshim në kod (në front-end apo back-end) testohet automatikisht, ndërtohet dhe publikohet në ambientin e prodhimit, duke e bërë zhvillimin më efikas dhe me më pak rreziqe për defekte në versionet live. Ky dizajn modular dhe i ndarë mirë garanton përfitime të shumëfishta në aspektin teknik dhe operacional. Ai siguron **shkallëzueshmëri horizontale** dhe **vertikale** të sistemit, mbështet testimin e njësiteve në mënyrë të pavarur, mundëson integritet të lehta me API të jashtme, dhe ofron bazën për vendosjen e praktikave moderne si caching, rate-limiting, auditing, logging, monitoring në kohë reale dhe aplikimin e standardeve të sigurisë si JWT, OAuth2 ose HTTPS.

Arkitektura është projektuar në mënyrë që të mbështesë jo vetëm nevojat e tanishme, por edhe të parashikojë zgjerime të ardhshme në drejtim të funksionalitetit, ngarkesës apo bashkëveprimit me sisteme të tjera, duke siguruar kështu një bazë solide për rritje të qëndrueshme dhe evoluim teknologjik.



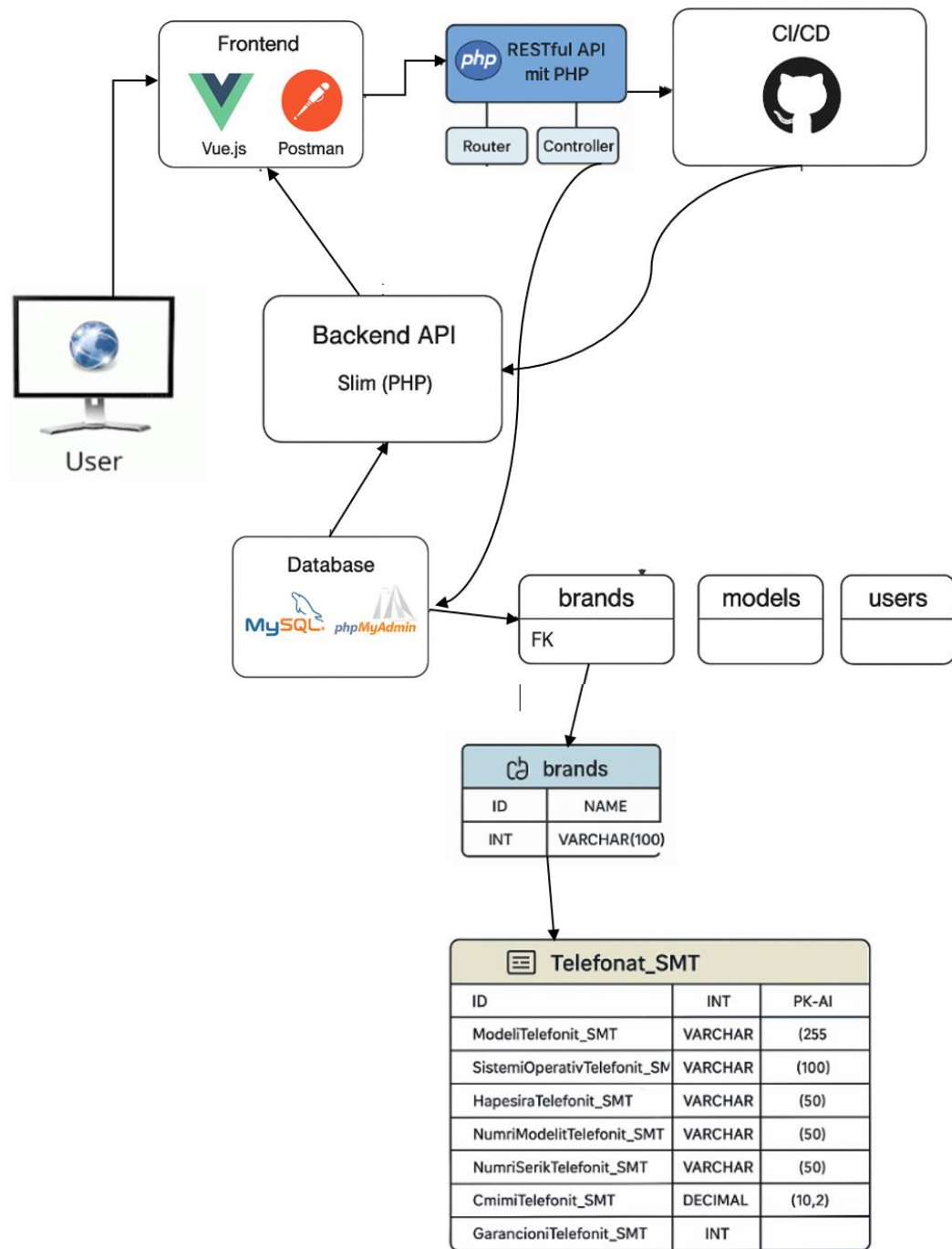


Figure 1. Dizajni Arkitekturës

### 3. CI/CD Pipeline (Git Actions)

Ky pipeline CI/CD me GitHub Actions është i ndërtuar për të automatizuar procesin e ndërtimit, testimit dhe përgatitjes së deploy-it të dy pjesëve kryesore të projektit: backend (me PHP Slim) dhe frontend (me Vue.js në folderin vSMT). Ai aktivizohet automatikisht në momentin që bëhet **push** ose **pull request** në branch-in main, duke siguruar që vetëm kodi i testuar dhe i kontrolluar të hyjë në këtë degë.

**Pjesa e Backend-it** funksionon në mjedisin ubuntu-latest. Fillimisht bëhet checkout i kodit nga repozitori, pastaj instalohet PHP version 8.2 me disa extensione të domosdoshme si mbstring, curl, json, dhe zip. Më pas përdoret Composer për të instaluar të gjitha varësitë e projektit në folderin ./slimapp. Pipeline vazhdon me ekzekutimin e testimeve automatike me PHPUnit nëse ekziston skedari phpunit.xml. Kjo siguron që backend-i funksionon si duhet para se të kalojë në fazat e mëtejshme.

**Pjesa e Frontend-it** gjithashtu ekzekutohet në ubuntu-latest. Bëhet checkout i kodit, pastaj instalohet Node.js version 18. Në folderin ./vSMT instalohet gjithë biblioteka dhe paketat e nevojshme për frontend-in duke përdorur npm install. Pas kësaj, pipeline ekzekuton testet e frontend-it me komandën npm run test, e cila është opsionale (nëse nuk ka test ose skedari i testit nuk është i përcaktuar, thjesht vazhdon). Në fund bëhet ndërtimi i versionit për prodhim me npm run build, duke përgatitur frontend-in për deploy.

Në kod ke edhe një pjesë të komentuar që i dedikohet **deploy-it në server** përmes SSH. Kjo pjesë do të përdorë kredencialet e siguruar si secrets (si IP e serverit, përdoruesi, çelësi SSH) për të hyrë në server dhe për të bërë git pull të kodit më të fundit në server, instalimin e varësive të prodhimit për backend dhe frontend, dhe për të rifilluar shërbimet e serverit (Apache ose PHP-FPM). Kjo do të mundësojë që versioni i ri i aplikacionit të jetë në prodhim pa ndërprerje manuale.

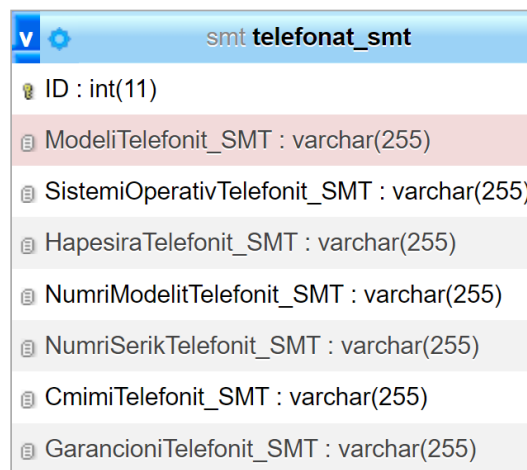
**Në total, ky pipeline ofron:**

- **Automatizim të plotë** nga marrja e kodit deri tek testimi dhe ndërtimi;
- **Kontroll cilësie** përmes testimeve automatike në backend dhe frontend;
- **Përgatitje për deploy** që mund të shtohet lehtësisht për automatizimin e shpërndarjes në server;
- **Siguri dhe qetësi** sepse çdo ndryshim në branch-in kryesor verifikohet dhe testohet automatikisht;
- **Modularitet** sepse backend dhe frontend trajtohen si punë të ndara brenda të njëjtit pipeline.

Ky është një shembull i mirë praktik që përmbush nevojat bazë për një pipeline CI/CD moderne në një projekt full-stack me PHP dhe JavaScript.

## 4. DIZAJNI DATABAZES

Për të krijuar një sistem për menaxhimin e telefonave, hapi i parë është krijimi i një databaze MySQL që do të mbajë të dhënat për telefonat. Kjo mund të arrihet duke përdorur komandat e MySQL për krijimin e databazës dhe tabelës përkatëse. Në këtë rast, kemi krijuar databazën **SMT** dhe një tabelë të quajtur **Telefonat\_SMT**, e cila do të ruajë informacionin e telefonave që do të menaxhohen. Kjo skemë është projektuar në mënyrë që të ofrojë mundësinë për ruajtjen e të dhënave të ndryshme që lidhen me telefonat, si modeli, sistemi operativ, hapësira, numri serik, çmimi dhe garancia, të cilat na shërbejnë për të kryer shërbime të ndryshme për menaxhimin e telefonave.



smt telefonat_smt	
ID	int(11)
ModeliTelefonit_SMT	varchar(255)
SistemiOperativTelefonit_SMT	varchar(255)
HapesiraTelefonit_SMT	varchar(255)
NumriModelitTelefonit_SMT	varchar(255)
NumriSerikTelefonit_SMT	varchar(255)
CmimiTelefonit_SMT	varchar(255)
GarancioniTelefonit_SMT	varchar(255)

Figure 2. Databaza

### Shpjegimi i kolonave të tabelës **Telefonat\_SMT**:

- **ID**: Kolona ID është çelësi kryesor (Primary Key, PK) i tabelës dhe është automatikisht në rritje (Auto Increment, AI). Ajo identifikon çdo telefon në mënyrë unike në tabelë.
- **ModeliTelefonit\_SMT**: Ky kolon ruan modelin e telefonit. Përdoret për të identifikuar telefonin në bazë të markës dhe versionit të tij (p.sh., "iPhone 13").
- **SistemiOperativTelefonit\_SMT**: Ky kolon ruan sistemin operativ që telefoni përdor (p.sh., "iOS", "Android").
- **HapesiraTelefonit\_SMT**: Ky kolon ruan hapësirën e telefonit në gigabajt (GB). Përdoret për të specifikuar sa hapësirë ka telefoni për ruajtjen e të dhënave.
- **NumriModelitTelefonit\_SMT**: Ky kolon ruan numrin e modelit të telefonit, që është i rëndësishëm për identifikimin e variacioneve të ndryshme të një modeli të caktuar.
- **NumriSerikTelefonit\_SMT**: Ky kolon ruan numrin serik të telefonit, i cili është një identifikues unik për çdo telefon dhe është i dobishëm për gjurmimin dhe garancinë.
- **CmimiTelefonit\_SMT**: Ky kolon ruan çmimin e telefonit në formatin decimal, përfshirë dy shifra pas presjes për saktësi të çmimit.
- **GarancioniTelefonit\_SMT**: Ky kolon ruan kohëzgjatjen e garancionit të telefonit në vite, i cili mund të jetë një informacion i rëndësishëm për menaxhimin e shërbimeve dhe politikave të kthimit.

## **5. IMPLEMENTIMI I APLIKACIONIT PËRMES POSTMAN-API BACK-end DHE FRONT-end**

### **5.1. RESTful API – Backend**

#### **Krijimi i një API me Slim PHP**

**Slim PHP** është një framework i thjeshtë dhe i shpejtë për zhvillimin e aplikacioneve web dhe API-ve. Ai ofron mundësinë për të krijuar endpoint-e të ndryshme që mund të kryejnë operacione të ndryshme mbi të dhënat e databazës në mënyrë të shpejtë dhe të sigurt. Slim është një framework i lehtë, por shumë fleksibël, i cili mund të përdoret për të ndërtuar aplikacione me kërkesa të thjeshta dhe të avancuara. Ai është një opsion i shkëlqyer për zhvillimin e API-ve RESTful që përdorin metodën e komunikimit HTTP për të transferuar të dhëna midis klientëve dhe serverëve.

**Në këtë projekt, API-ja do të përfshijë një seri endpoint-esh që mundësojnë operacione të ndryshme si:**

#### **1. Krijimi i një Telefoni të Ri**

Ky endpoint do të mundësojë që përdoruesit të shtojnë telefonat e rinj në databazë. Përdoruesi do të dërgojë informacionin e telefonit (p.sh., modeli, sistemi operativ, hapësira, numri serik, etj.) në formën e një kërkesë HTTP POST, dhe API-ja do ta ruajë këtë informacion në tabelën Telefonat\_SMT. Kjo mundëson që përdoruesit të shtojnë telefonë të rinj me informacion të plotë dhe të saktë.

#### **2. Leximi i të Dhënave të Telefonëve**

Ky endpoint mundëson që përdoruesit të lexojnë të dhënat e telefonëve që janë tashmë të ruajtura në databazë. Përdoruesi mund të kërkojë një listë të të gjitha telefonave që janë regjistruar në sistem, ose mund të kërkojë të dhëna specifike për një telefon të caktuar duke përdorur ID-në e tij. Ky operacion është i rëndësishëm sepse mundëson shikimin e informacionit në kohë reale dhe siguron që përdoruesi të ketë mundësi të monitorojë dhe menaxhojë telefonat që janë regjistruar në sistem.

#### **3. Përditësimi i të Dhënave të Telefonëve**

Përdoruesit mund të kenë nevojë të përditësojnë të dhënat e telefonëve të regjistruar. Kjo mund të ndodhë për arsye të ndryshme, si përditësimi i çmimit të telefonit, ndryshimi i modelit të telefonit, ose ndonjë informacion tjetër që mund të ndryshojë me kalimin e kohës (p.sh., përditësimi i informacionit të garancisë). Ky endpoint mundëson përditësimin e të dhënave ekzistuese në databazë, dhe gjithashtu siguron që të dhënat të mbeten të sakta dhe të freskëta.

#### **4. Fshirja e një Telefoni nga Databaza**

Ky endpoint mundëson fshirjen e një telefoni të caktuar nga databaza, nëse nuk është më i nevojshëm ose për shkak të ndonjë arsye tjetër që lidhet me menaxhimin e telefonave në sistem. Përdoruesi mund të kërkojë fshirjen e telefonit nëpërmjet ID-së së tij, dhe API-ja do të sigurojë që të dhënat përkatëse të fshihen nga tabelat përkatëse të databazës.

#### **Funksionalitetet e API-së dhe Interaksioni me Databazën**

Për të siguruar që API-ja funksionon në mënyrë të përshtatshme, është e rëndësishme që ajo të jetë e ndërtuar në një mënyrë që mundëson operacione të sakta dhe efikase mbi të dhënat e databazës. Kjo do të

thotë që çdo kërkesë e dërguar në API duhet të përpunojë informacionin në mënyrë të saktë dhe të sigurojë që nuk ka gabime në ruajtjen e të dhënave.

## Siguria dhe Autentifikimi i API-së

Në krijimin e një API-je, është shumë e rëndësishme të merren parasysh aspekte të sigurisë, përfshirë autentifikimin dhe autorizimin. Ky aspekt është veçanërisht i rëndësishëm për të siguruar që vetëm përdoruesit e autorizuar të mund të bëjnë ndryshime në të dhënat e telefonave. Përdorimi i mekanizmave të autentifikimit si JSON Web Tokens (JWT) ose OAuth mund të ndihmojë në sigurinë e API-së dhe në mbrojtjen e të dhënave sensitive. Përveç kësaj, është e rëndësishme të merret parasysh përdorimi i metodave të sigurisë si HTTPS për të siguruar që të dhënat që kalojnë ndërmjet klientit dhe serverit të jenë të mbrojtura nga ndonjë sulm i mundshëm, si p.sh. ndërhyrja e të dhënave (man-in-the-middle attacks).

Pas krijimit të databazës dhe tabelës, hapi i natyrshëm tjetër është krijimi i një API-je për të mundësuar kryerjen e operacioneve të ndryshme mbi të dhënat që ruhen në tabelën **Telefonat\_SMT**. Ky API është një ndërfaqe që lejon komunikimin midis aplikacionit frontend dhe databazës përmes protokolleve të thjeshta të HTTP. API-ja mundëson që të dhënat e telefonave të jenë të aksesueshme dhe të menaxhueshme nga përdoruesit, si dhe mundëson operacione të ndryshme si: **Create, Read, Update dhe Delete** - të njohura ndryshe si **CRUD**, e të dhënave që ndodhen në databazën tonë. Postman është një aplikacion i cili përdoret për back-end. Ky aplikacion na ofron mundësi shërbyese shumë të mira i cili na ndimmon të paraqesim të dhënat që kemi në databazë, nëse i kërkojmë të gjitha na paraqet të gjitha, nëse kërkojmë një të dhënë na shfaq një të dhënë, na shërben për të shtuar të dhëna, për të modifikuar të dhëna si dhe për të fshirë të dhënat që dëshirojmë.

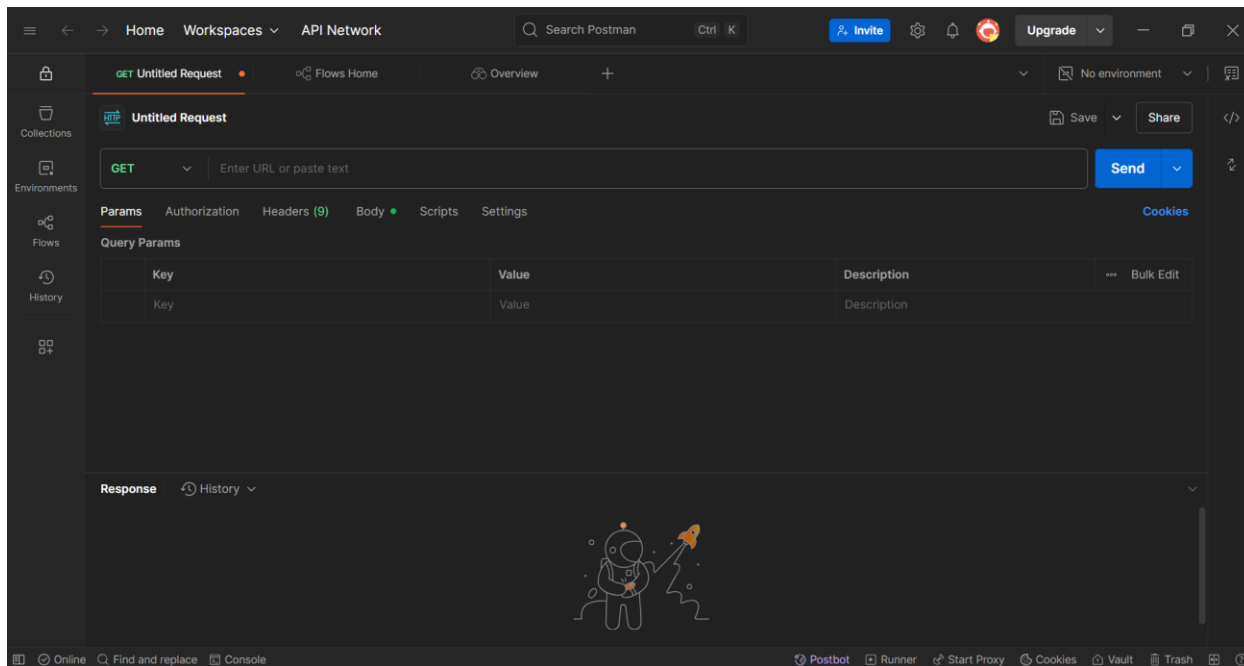


Figure 3. Paraqitja e Postman – Backend

### 5.1.1. Paraqitja e të gjitha të dhënave

Në këtë hap përdorim metodën **GET**, e cila është e dizajnuar për të **lexuar të dhënat** nga serveri.

Në aplikacionin **Postman**, pasi hapim një dritare të re për testim:

- Zgjedhim metodën GET nga menyja e metodave HTTP.
- Tek fusha e URL-së shkruajmë endpoint-in e krijuar në backend, që për shembull mund të jetë:

**http://localhost/sliampppp/public/telefonat**

- Me këtë kërkesë, ne kërkojmë nga API-ja që të **na kthejë të gjitha të dhënat** ekzistuese në tabelën Telefonat\_SMT.
- Pasi të klikojmë **Send**, Postman komunikon me serverin dhe na paraqet në seksionin e përgjigjes të gjitha rreshtat ekzistues në formatin **JSON**, që përfshin të dhëna të tilla si:
  - ID
  - Modeli i telefonit
  - Sistemi operativ
  - Hapësira
  - Numri i modelit
  - Numri serik
  - Çmimi
  - Garancioni
- Kjo na ndihmon të sigurohemi që API-ja është funksionale dhe që të dhënat po lexohen saktësisht nga databaza.

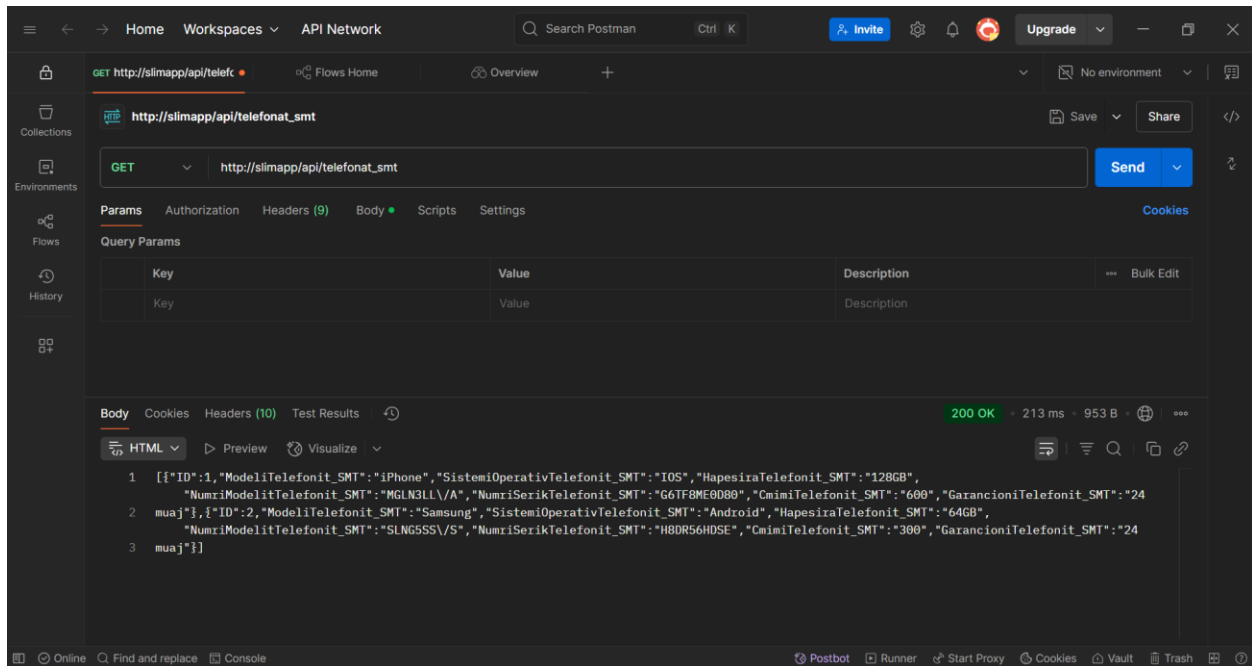


Figure 4. Paraqitja e të gjitha të dhënave – Backend

### 5.1.2. Paraqitja e një të dhëne të caktuar sipas ID-së

Përdorimi i metodës **GET** në këtë rast shërben për të marrë **vetëm një të dhënë specifike**, bazuar në **ID-në unike** të saj.

- Në Postman, zgjedhim metodën GET dhe në URL vendosim endpoint-in së bashku me ID-në e të dhënës që dëshirojmë ta shohim, si p.sh.:

**<http://localhost/sliampppp/public/telefonat/3>**

- Kjo kërkesë i thotë API-së të kërkojë në databazë për rreshtin që ka **ID = 3**, dhe të na e kthejë atë në përgjigje.
- Rezultati paraqitet në format JSON dhe përmban të gjitha fushat e plotësuara për atë telefon specifik.
- Ky funksionalitet është veçanërisht i dobishëm kur duam të konsultohemi vetëm me një produkt pa pasur nevojë të shkarkojmë të gjitha të dhënat.

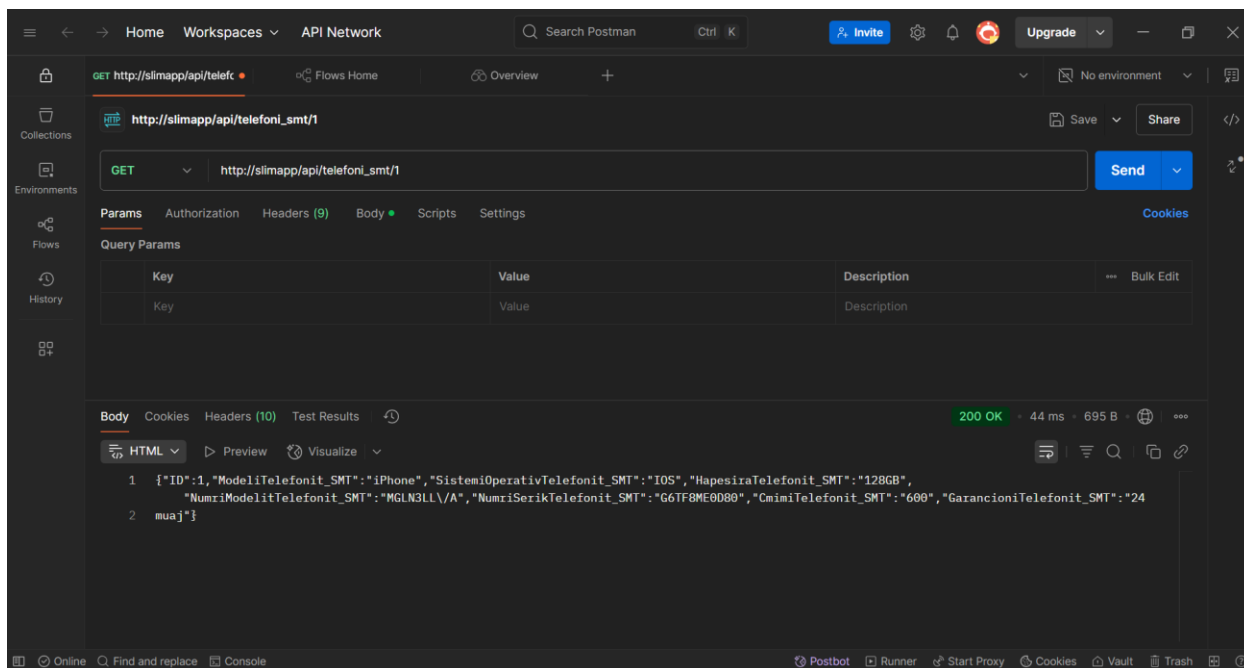


Figure 5. Paraqitja e një të dhëne – Backend

### 5.1.3. Shtimi i të dhënave

Ky hap përfshin **krijimin e të dhënave të reja në databazë**, përmes metodës HTTP **POST**.

- Në Postman, zgjedhim metodën POST.
- Vendosim URL-në për endpoint-in që pranon shtimin e të dhënave, si p.sh.:

**<http://localhost/sliampppp/public/telefonat/add>**

- Më pas, klikojmë në skedën **Body**, zgjedhim opsionin **raw**, dhe si format zgjedhim **JSON**.
- Në këtë seksion shtojmë një strukturë JSON që përmban të dhënat që dëshirojmë të shtojmë. P.sh.:

```
{  
  
  "ModeliTelefonit_SMT": "Samsung Galaxy S23",  
  
  "SistemiOperativTelefonit_SMT": "Android",  
  
  "HapesiraTelefonit_SMT": "256GB",  
  
  "NumriModelitTelefonit_SMT": "SM-S911B",  
  
  "NumriSerikTelefonit_SMT": "R5CT1234567",  
  
  "CmimiTelefonit_SMT": "850",  
  
  "GarancioniTelefonit_SMT": "24 muaj"  
}
```

Pas klikimit të **Send**, nëse gjithçka është konfiguruar siç duhet, do të marrim një përgjigje që konfirmon se të dhënat janë shtuar me sukses në databazë.

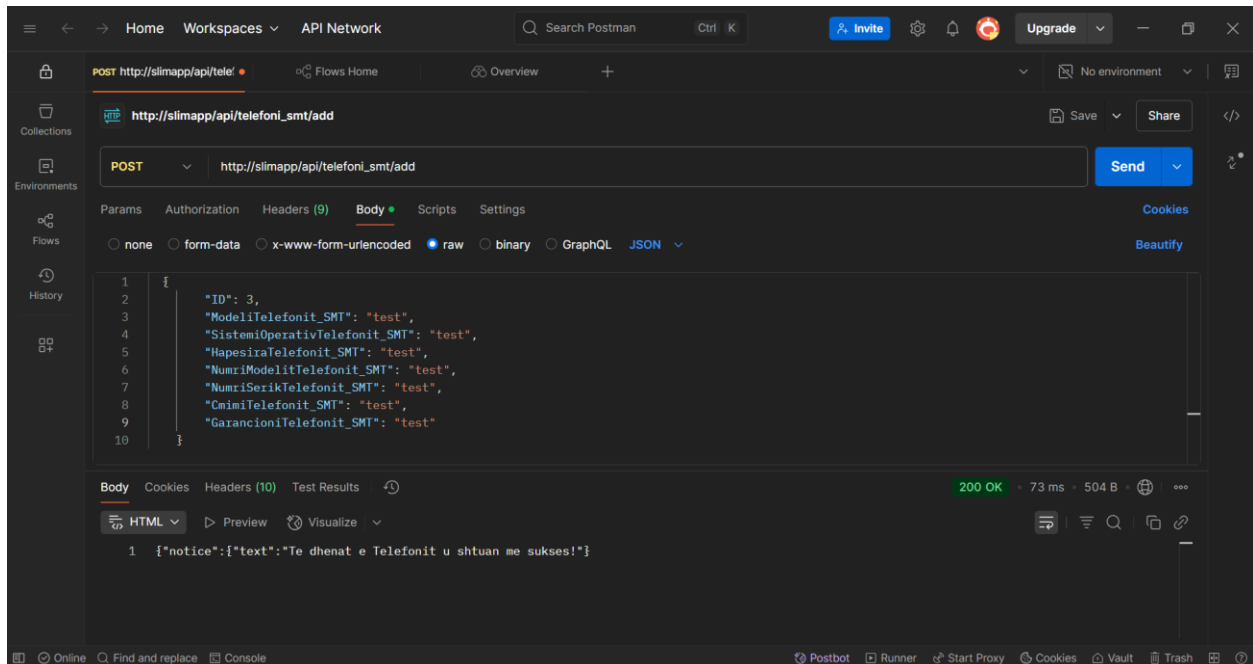


Figure 6. Shtimi i të dhënave – Backend



#### 5.1.4. Modifikimi i të dhënave

Për përditësimin e të dhënave ekzistuese përdorim metodën **PUT**, e cila është e destinuar për të modifikuar një rresht ekzistues në databazë.

- Në Postman, zgjedhim metodën PUT.
- Në URL vendosim endpoint-in për përditësim së bashku me ID-në e të dhënës që duam të modifikojmë. Shembull:

**<http://localhost/sliampppp/public/telefonat/update/3>**

- ☐ Shkojmë sërish në skedën **Body**, zgjedhim **raw**, dhe si format përdorim **JSON**.
- ☐ Vendosim të dhënat e reja që dëshirojmë t'i ruajmë, duke përfshirë fushat e modifikuara:

```
{  
  "ModeliTelefonit_SMT": "Samsung Galaxy S24 Ultra",  
  "SistemiOperativTelefonit_SMT": "Android 14",  
  "HapesiraTelefonit_SMT": "512GB",  
  "NumriModelitTelefonit_SMT": "SM-S928B",  
  "NumriSerikTelefonit_SMT": "R5CT9876543",  
  "CmimiTelefonit_SMT": "1050",  
  "GarancioniTelefonit_SMT": "36 muaj"  
}
```

Pas klikimit të **Send**, nëse gjithçka ka kaluar me sukses, API-ja do të kthejë një mesazh konfirmues që rreshti me ID-në e caktuar është përditësuar me sukses.

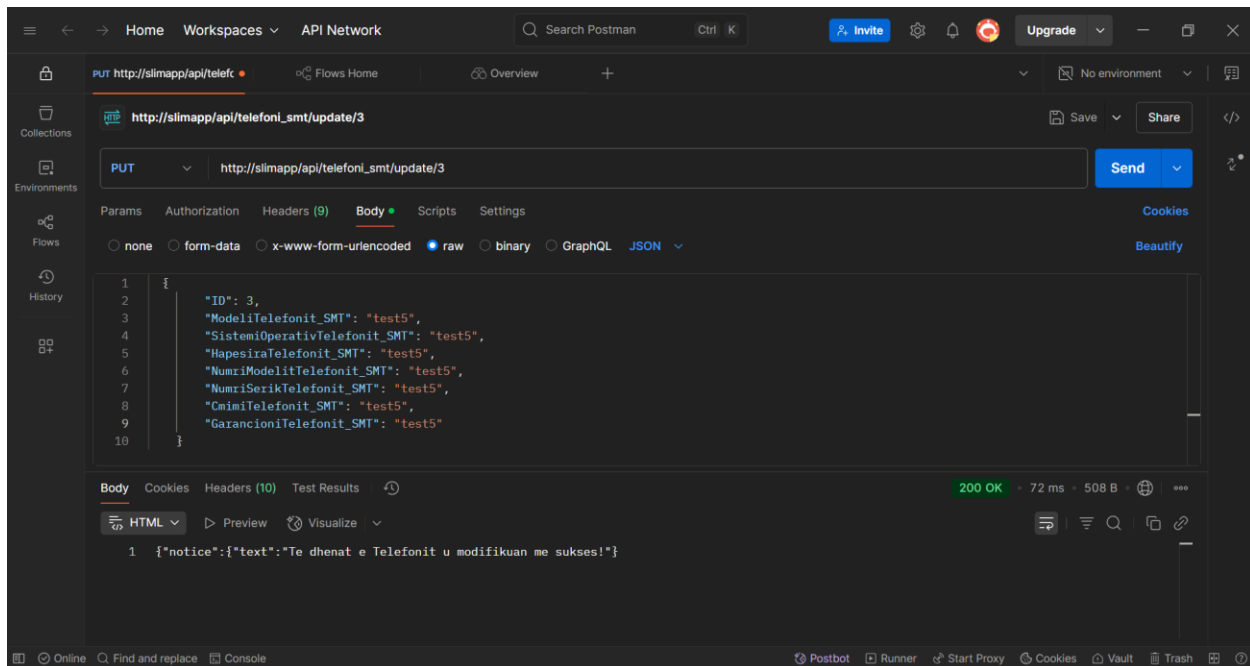


Figure 7. Modifikimi i të dhënave – Backend

### 5.1.5. Fshirja e të dhënave

Ky funksionalitet shërben për të **fshirë të dhëna nga databaza**, duke përdorur metodën **DELETE**.

- Në Postman, zgjedhim metodën DELETE.
- Në fushën e URL-së shkruajmë endpoint-in për fshirje, bashkë me ID-në e të dhënës që dëshirojmë të fshijmë. Shembull:

**<http://localhost/sliampppp/public/telefonat/delete/3>**

- Pas dërgimit të kërkesës, API-ja do të përpunojë kërkesën dhe do të ekzekutojë komandën për të fshirë të dhënën me atë ID specifike nga databaza.
- Në përgjigje, Postman do të shfaqë një mesazh që konfirmon se të dhënat janë fshirë me sukses, duke siguruar që operacioni është kryer siç duhet.

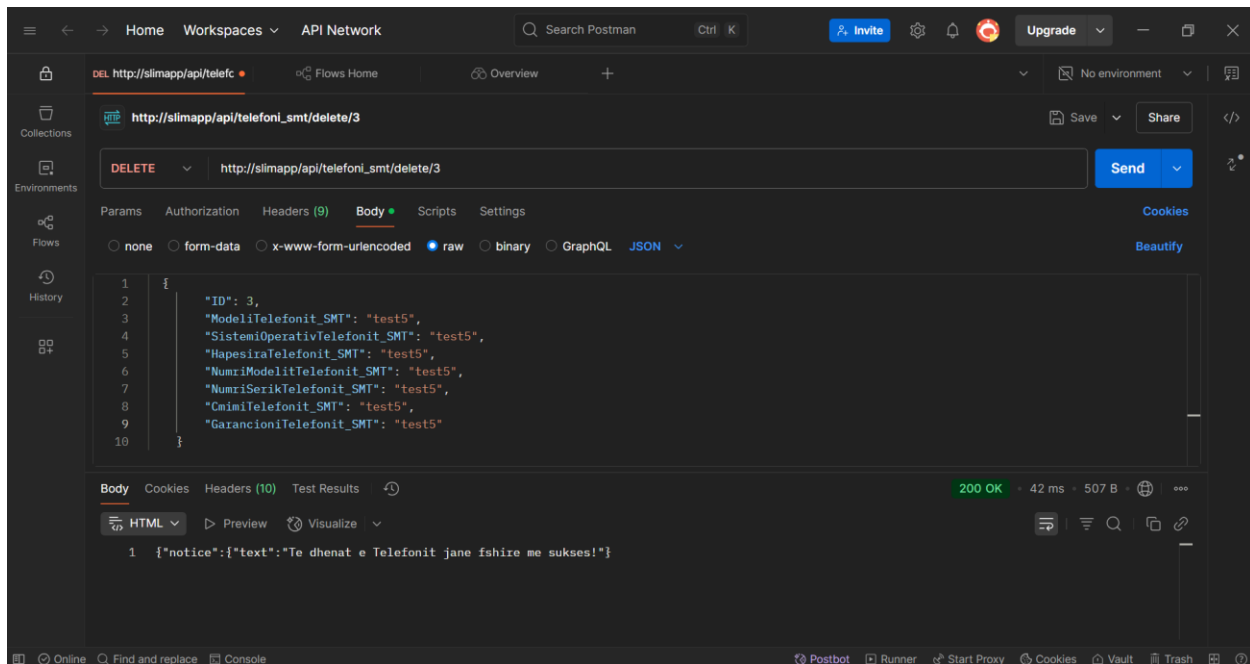


Figure 8. Fshirja e të dhënave – Backend

Në këtë mënyrë, përmes ndërtimit të një **RESTful API-je me Slim PHP** dhe testimit me **Postman**, kemi realizuar të gjithë ciklin e operacioneve **CRUD** ndaj të dhënave në databazën SMT, duke mundësuar ndërveprim të plotë midis përdoruesit, aplikacionit dhe serverit.

## 5.2. RESTful API – Frontend

Pasi kemi perfunduar pjesën e Back-end, kemi bërë edhe pjesën e Front-end, ku në këtë dritare shohim që na janë paraqitur të gjitha të dhënat. Pasi kemi modifikuar të dhënat kërkuese kemi shkruar në folderin vSMT dhe kemi shkruar në cmd, kemi shkruar `npm run dev` na ka paraqitur `localhost:8081`, kemi marrur këtë link dhe kemi vendosur në google dhe na janë paraqitur të dhënat që kemi në database, kjo na shërben për të shtuar të dhënat, për të modifikuar si dhe për të fshirë të dhënat. Frontendi është ndërtuar në **Vue.js**, një framework i fuqishëm JavaScript për ndërtimin e ndërfaqeve të përdoruesit reaktive dhe dinamike. Ky frontend bën thirrje të drejtpërdrejta API-ve të backend-it (REST API), duke përdorur **metoda HTTP** si GET, POST, PUT dhe DELETE. Këto thirrje lejojnë ndërveprimin e plotë të përdoruesit me të dhënat në databazë, përmes një ndërfaqeje të thjeshtë por funksionale.

## Sistemi per Menaxhimin e Telefonave (SMT)

Sheno Modelin e Telefonit

ModeliTelefonit_SMT	SistemiOperativTelefonit_SMT	HapesiraTelefonit_SMT	NumriModelitTelefonit_SMT	NumriSerikTelefonit_SMT	CmimiTelefonit_SMT	GarancioniTelefonit_SMT	
iPhone	IOS	128GB	MGLN3LL/A	G6TF8ME0D80	600	24 muaj	<div>Modifiko</div> <div>Fshije</div>
Samsung	Android	64GB	SLNG5SS/S	H8DR56HDSE	300	24 muaj	<div>Modifiko</div> <div>Fshije</div>
Huawei	Android	32GB	HFV8SB	TGFS96	100	HGR6JNB	<div>Modifiko</div> <div>Fshije</div>
Blackberry	Android	32GB	Y9JN6DC	OJMN6R	150	6muaj	<div>Modifiko</div> <div>Fshije</div>

Figure 9. Paraqitja e të dhënave – Frontend

### 5.2.1. Paraqitja e të gjitha të dhënave

**Qëllimi:** Të shfaqen të gjithë telefonat e regjistruar në databazën SMT në një pamje të vetme.

**Si funksionon:**

- Kur ekzekutohet komanda `npm run dev` në folderin vSMT, aktivizohet serveri lokal i frontend-it dhe aplikacioni bëhet i qasshëm në `http://localhost:8081`.
- Në hapjen e faqes, komponenti kryesor Vue (App.vue ose Home.vue) bën një **thirrje HTTP GET** për të tërhequr të gjitha të dhënat nga rruta backend, zakonisht:

**GET `http://localhost:8081/telefonat`**

- Të dhënat e marra (në JSON) ruhen në një `data()` array, si telefonat, dhe shfaqen në formë **tabele** ose **kartelash** (cards) në UI duke përdorur `v-for`.

**Përfitimi për përdoruesin:**

- E shoh në mënyrë të strukturuar çdo telefon të ruajtur me detajet: modeli, sistemi operativ, hapësira, çmimi, etj.
- Ka opsione të dukshme për modifikim apo fshirje për secilën rresht të dhënash.

## Sistemi per Menaxhimin e Telefonave (SMT)

Sheno Modelin e Telefonit

ModeliTelefonit_SMT	SistemiOperativTelefonit_SMT	HapesiraTelefonit_SMT	NumriModelitTelefonit_SMT	NumriSerikTelefonit_SMT	CmimiTelefonit_SMT	GarancioniTelefonit_SMT	
iPhone	IOS	128GB	MGLN3LL/A	G6TF8ME0D80	600	24 muaj	Modifiko Fshije
Samsung	Android	64GB	SLNG5SS/S	H8DR56HDSE	300	24 muaj	Modifiko Fshije
Huawei	Android	32GB	HFV8SB	TGFS96	100	HGR6JNB	Modifiko Fshije
Blackberry	Android	32GB	Y9JN6DC	OJMN6R	150	6muaj	Modifiko Fshije

Figure 10. Paraqitja e të dhënave – Frontend

### 5.2.2. Kërkimi i të dhënave

**Qëllimi:** Të mundësohet kërkimi specifik i një telefoni sipas modelit.

**Si funksionon:**

- Përdoruesi fut tekst në një **fushë input** (placeholder: “*Shkruaj modelin e telefonit*”).
- Kur përdoruesi shkruan një model, ndodh një **filterim në frontend** (në memorien lokale), ku përdoret një direktivë v-model e lidhur me një funksion computed ose watcher që filtron array-in telefonat bazuar në emrin e modelit.
- Opsionalisht, mund të bëhet edhe kërkim përmes një **API-thirrjeje të veçantë**:

**GET http://localhost:8081/telefonat/{id}**

**Përfitimi për përdoruesin:**

- Redukton kohën e kërkimit të një telefoni të caktuar.
- Jep informacion të saktë për modelin: sistemin operativ, numrin serik, çmimin, etj.

vSMT
Ballina
Rreth nesh
Shto te dhënat e Telefonit

### Sistemi per Menaxhimin e Telefonave (SMT)

ModeliTelefonit_SMT	SistemiOperativTelefonit_SMT	HapesiraTelefonit_SMT	NumriModelitTelefonit_SMT	NumriSerikTelefonit_SMT	CmimiTelefonit_SMT	GarancioniTelefonit_SMT	
Samsung	Android	64GB	SLNG5SS/S	H8DR56HDSE	300	24 muaj	<div>Modifiko</div> <div>Fshije</div>

Figure 11. Kërkimi i të dhënave – Frontend

### 5.2.3. Shtimi i të dhënave

**Qëllimi:** T’i mundësohet përdoruesit që të shtojë një telefon të ri në databazë përmes ndërfaqes grafike.

**Si funksionon:**

- Klikohet butoni “**Shto të dhënat e telefonit**”, që hap një **formë HTML** e cila përmban input fields për të gjitha fushat:
  - ModeliTelefonit\_SMT
  - SistemiOperativTelefonit\_SMT
  - HapesiraTelefonit\_SMT
  - NumriModelitTelefonit\_SMT
  - NumriSerikTelefonit\_SMT
  - CmimiTelefonit\_SMT
  - GarancioniTelefonit\_SMT
- Përdoruesi plotëson formën dhe klikon butonin “**Shto**”, që ekzekuton një **thirrje HTTP POST**:

```

axios.post('http://localhost:8081/telefonat/add', {

    modeli: this.modeli,

    sistemiOperativ: this.sistemiOperativ,

    ...

})

```

- Nëse operacioni është i suksesshëm, shfaqet një alert ose toast: “*Të dhënat u shtuan me sukses!*”

**Përfitimi për përdoruesin:**

- Mundësi për të regjistruar pajisje të reja.
- Krijon një ndërveprim të thjeshtë dhe miqësor për të dhënat e reja pa pasur nevojë të hyjë në databazë manualisht.

## Shto te dhenat e Telefonit

Informacionet e Modelit Telefonit:

**ModeliTelefonit\_SMT**

**HapesiraTelefonit\_SMT**

**CmimiTelefonit\_SMT**

Informacionet e Sistemit Operativ te Telefonit:

**SistemiOperativTelefonit\_SMT**

**NumriModelitTelefonit\_SMT**

**NumriSerikTelefonit\_SMT**

**GarancioniTelefonit\_SMT**

Shto

Figure 12. Shtimi i të dhënave -Frontend

### 5.2.4. Modifikimi i të dhënave

**Qëllimi:** Të mundësohet përditësimi i të dhënave të një telefoni ekzistues.

**Si funksionon:**

- Përdoruesi klikon mbi butonin "**Modifiko**" pranë një telefoni.
- Hapet një **formë modifikimi** me v-model të lidhura me vlerat ekzistuese të telefonit.
- Përdoruesi ndryshon vlerat dhe klikojnë "Modifiko", që bën një **thirrje HTTP PUT**:

```
axios.put(`http://localhost:8081/telefonat/update/${id}`, {  
  modeli: this.modeli,  
  sistemiOperativ: this.sistemiOperativ,  
  ...  
})
```

- Backend-i përditëson të dhënat, dhe pas suksesit, frontend rifreskon listën e telefonave për të reflektuar ndryshimet.

### Përfitimi për përdoruesin:

- Jep kontroll të plotë për korrigjimin e gabimeve ose përditësimin e informacionit teknologjik të një pajisjeje.

#### Modifiko te dhenat e Telefonit

Informacionet e Modelit Telefonit:

ModeliTelefonit\_SMT

test1

HapesiraTelefonit\_SMT

test1

CmimiTelefonit\_SMT

test1

Informacionet e Sistemit Operativ te Telefonit:

SistemiOperativTelefonit\_SMT

test1

NumriModelitTelefonit\_SMT

test1

NumriSerikTelefonit\_SMT

test1

GarancioniTelefonit\_SMT

test1

Modifiko

Figure 13. Modifikimi i të dhënave – Frontend

### 5.2.5. Fshirja e të dhënave

**Qëllimi:** Të lejohet heqja e telefonave të padëshiruar ose të vjetruar nga databaza.

#### Si funksionon:

- Përdoruesi klikon butonin “**Fshij**” pranë telefonit që dëshiron të heqë.
- Shfaqet një confirm() ose alert për të siguruar që veprimi është i qëllimshëm.
- Nëse konfirmohet, bëhet një **thirrje HTTP DELETE**:

**`axios.delete('http://localhost:8081/telefonat/delete/${id}')`**

- Pas fshirjes së suksesshme, lista e telefonave rifreskohet automatikisht, dhe shfaqet një alert: “*Të dhënat e telefonit janë fshirë me sukses.*”

### Përfitimi për përdoruesin:

- Siguron menaxhim efikas të inventarit të telefonave.
- Pastron të dhënat e panevojshme duke ruajtur integritetin e databazës.



vSMT

Ballina

Rreth nesh

Shto te dhenat e Telefonit

Te dhenat e Telefonit u Fshine me sukses!

Sistemi per Menaxhimin e Telefonave (SMT)

Sheno Modelin e Telefonit

ModeliTelefonit_SMT	SistemiOperativTelefonit_SMT	HapesiraTelefonit_SMT	NumriModelitTelefonit_SMT	NumriSerikTelefonit_SMT	CmimiTelefonit_SMT	GarancioniTelefonit_SMT	
IPhone	IOS	128GB	MGLN3LL/A	G6TF8ME0D80	600	24 muaj	<div>Modifiko</div> <div>Fshije</div>
Samsung	Android	64GB	SLNG5SS/S	H8DR56HDSE	300	24 muaj	<div>Modifiko</div> <div>Fshije</div>
Huawei	Android	32GB	HFV8SB	TGFS96	100	HGR6JNB	<div>Modifiko</div> <div>Fshije</div>
Blackberry	Android	32GB	Y9JN6DC	OJMNG6R	150	6muaj	<div>Modifiko</div> <div>Fshije</div>

Figure 14. Fshirja e të dhënave – Frontend

Ky frontend e ndërlidh me elegancë dhe lehtësi përdoruesin me të dhënat e ruajtura në databazë, duke shfrytëzuar fuqinë e RESTful API-ve dhe Vue.js. Ai është ndërtuar për të qenë interaktiv, i thjeshtë për t’u përdorur, dhe i përshtatur me praktikatat moderne të zhvillimit të aplikacioneve.

## 6. CACHING AND PERFORMANCE OPTIMIZATION, MESSAGE QUEUES AND EVENT-DRIVEN ARCHITECTURE, FAULT TOLERANCE, HIGH AVAILABILITY, AND RESILIENCE

### 6.1. Caching and Performance Optimization

Një komponent kyç për caching në këtë projekt është **Redis** – një sistem memorie in-memory i tipit key-value, i cili është integruar për të ruajtur të dhëna të përkohshme që përdoren shpesh nga klientët ose backend-i.

#### Shembuj konkret të përdorimit:

- Në skedarin `test_redis.php`, ilustrohet një shembull bazik i ruajtjes së një vlere (`test_key`) dhe leximi i saj nga Redis.
- Në implementime reale, të tilla si pyetjet ndaj databazës MySQL për lista të telefonave, këto të dhëna mund të ruhen në Redis për të shmangur pyetjet e përsëritura të ngadalta.

#### Përfitime direkte:

- Ulje drastike e **kohës së përgjigjes** në kërkesa të përsëritura.
- Reduktim i **ngarkesës në databazë** për operacionet `SELECT`.
- Përmirësim i **efikasitetit të rrjetit**, pasi të dhënat janë menjëherë të disponueshme në memorien RAM përmes Redis.

#### Optimizimi i Performancës në Nivel Aplikacioni

Përveç caching, projekti ka implementuar një sërë teknikash për optimizimin e performancës në kod dhe në nivelin e shërbimeve.

- Përmes përdorimit të **Postman** janë testuar skenarë me ngarkesë të lartë për të matur kohën e përgjigjes dhe për të identifikuar pikat e ngadalësimit.
- Redis ndihmon që këto testime të japin rezultate më të mira duke shmangur vonesat e panevojshme nga databaza.

Zbatimi i caching me Redis dhe struktura e ndërtuar në Slim PHP përbëjnë një bazë të fortë për ndërtimin e një sistemi performant. Këto teknika jo vetëm që përmirësojnë përvojën e përdoruesit final, por gjithashtu kontribuojnë në një arkitekturë më të shkallëzueshme dhe më efikase për përpunimin e kërkesave.

### 6.2. Message Queues and Event-Driven Architecture

Arkitektura e Bazuar në Ngjarje (Event-Driven Architecture – EDA) është një paradigmë moderne për ndërtimin e sistemeve të shpërndara dhe fleksibile, ku komponentët komunikojnë përmes ngjarjeve (events) në vend të thirrjeve direkte (direct calls). Ky model rrit ndjeshëm modularitetin, asinkronizimin dhe shkallëzueshmërinë e aplikacioneve moderne.

Në projektin **SMT (Sistemi për Menaxhimin e Telefonave)**, është integruar një model i thjeshtë, por funksional i kësaj arkitekture, përmes përdorimit të **Redis** si sistem i thjeshtë i **Message Queue**, dhe skriptit `QueueWorker.php` për përpunimin asinkron të ngjarjeve.

### Komponentët kryesorë të Event-Driven Architecture në SMT

**a. Redis si Message Queue:** Redis është përdorur për të menaxhuar një `task_queue`, në të cilën aplikacioni shton ngjarje ose detyra që kërkojnë përpunim të mëvonshëm. Kjo qasje ndan në mënyrë të qartë logjikën e ekzekutimit nga logjika e përgjigjes së API-së.

#### Përfitimet:

- Asinkronizim i ngjarjeve (eventual consistency)
- Izolim i logjikës së ngarkuar nga rrjedha kryesore e aplikacionit
- Lehtësi për skalim horizontal me shumë punëtorë (workers)

**b. Queue Worker (QueueWorker.php):** Ky komponent funksionon si **konsumer** i radhës (queue), duke marrë ngjarjet (detyrat) dhe duke i përpunuar ato një nga një në mënyrë të pavarur.

```
while (true) {  
  
    $job = $redis->popFromQueue('task_queue');  
  
    if ($job) {  
  
        $data = json_decode($job, true);  
  
        // përpunimi i detyrës  
  
    } else {  
  
        sleep(1); // pushim në mungesë detyre  
  
    }  
  
}
```

### Shembuj praktikë të përdorimit në SMT

Një shembull tipik: kur shtohet një telefon i ri në databazë përmes një API-je, në vend që të përpunohen të gjitha aspektet menjëherë (si validimi, ruajtja, gjenerimi i log-ut, etj.), një **event** mund të dërgohet në `task_queue`, dhe `QueueWorker.php` e përpunon atë më vonë.

Ky model ndihmon që API të jetë më i **shpejtë** dhe më i **qëndrueshëm**, edhe kur ka ngarkesë të lartë.

#### Përfitime të Arkitekturës me Event në SMT:

Përfitim	Përshkrim
Asinkronizim i ngjarjeve	Lejon përpunimin e detyrave pa bllokuar përdoruesin
Reduktim i kohës së përgjigjes	API përgjigjet menjëherë, ndërsa detyrat përpunohen në sfond
Shkallëzueshmëri	Mund të shtohen më shumë QueueWorker për të përballuar më shumë ngarkesë
Modularitet	Logjika e detyrave është e ndarë nga logjika e shërbimit kryesor

Integrimi i **Redis Queue** dhe `QueueWorker.php` përfaqëson një implementim fillestar të një arkitekture të bazuar në ngjarje brenda sistemit SMT. Ky stil arkitekturor ofron përfitime të dukshme në performancë, modularitet dhe qëndrueshmëri, duke përgatitur sistemin për zgjerime më të mëdha në të ardhmen.

### 6.3. Fault Tolerance

Fault tolerance i referohet aftësisë së një sistemi për të vazhduar të funksionojë normalisht, edhe kur ndodhin gabime (p.sh., dështime të komponentëve të brendshëm, shërbime të jashtme, etj.).

Në projektin tim, janë zbatuar disa mekanizma që kontribuojnë në Fault Tolerance:

- QueueWorker me Retry logjikë të thjeshtë: Nëse nuk ka punë apo një task nuk është valid, worker-i nuk dështon menjëherë, por përdor një loop me sleep (1) për të qëndruar aktiv dhe i gatshëm për detyra të reja.
- Kontrolli i lidhjes me Redis përmes klasës `RedisClient`: Nëse Redis nuk është i disponueshëm, sistemi nuk do të thyejë direkt rrjedhën e aplikacionit kryesor, por është e mundur të përdorësh `try-catch` për të kapur gabimet në versionin e ardhshëm.
- Kontrolli i CORS dhe Headers në `index.php` ndihmon në shmangien e dështimeve të papritura në komunikimin e frontend-it me backend-in, duke e bërë API më të qëndrueshme.

#### Shembull praktikë:

- Nëse Redis ndalon përkohësisht, sistemi ruan stabilitetin e përgjithshëm, duke mos e bllokuar API-në.
- QueueWorker vazhdon të funksionojë pa ndërprerje, duke menaxhuar gabimet me qasjen e qëndrueshme (`while-true loop`)

### 6.4. High Availability

High Availability (HA) ka të bëjë me aftësinë e një sistemi që të qëndrojë aktiv dhe i përdorshëm për shumicën e kohës, me minimum downtime.

Edhe pse SMT është një sistem në fazën zhvillimore, janë marrë disa hapa të qartë drejt HA:

- **Përdorimi i Redis** si komponent i pavarur, i cili mund të vendoset në një server të dedikuar dhe të lidhet me më shumë instanca aplikacioni.
- Aplikacioni është i ndarë në komponentë të pavarur (API Slim, databazë MySQL, Redis Worker), çka lejon rifillim selektiv të komponenteve në rast dështimi.

#### **Shembull praktikë:**

- Nëse aplikacioni Slim dështoi për ndonjë arsye, `QueueWorker` mund të vazhdojë të përpunojë detyra të mbetura.
- Ndërtimi modular e bën të mundur vendosjen në ambiente të cloud-it me load balancers, për një implementim të vërtetë HA në të ardhmen.

## **6.5. Resilience**

Resilience është aftësia e sistemit për t'u rikuperuar dhe për t'u kthyer në një gjendje të qëndrueshme pas një dështimi apo ndërprerjeje.

#### **Zbatimi në SMT:**

Resilience është ndërtuar përmes kombinimit të:

- Loop-it të qëndrueshëm në `QueueWorker.php`, që nuk e ndalon funksionimin edhe në mungesë të detyrave.
- Përdorimi i një sistemi të jashtëm (Redis) që mund të ringrihet më vete pa afektuar logjikën e aplikacionit.
- Kod i organizuar dhe i lehtë për riparim, i ndarë në module ku çdo komponent mund të testohen apo rifillohen në mënyrë të pavarur.

#### **Shembull praktikë:**

- Në rast se Redis dështon, punëtori (`QueueWorker`) mund të rifillojë automatikisht kur Redis rikthehet, pa ndërhyrje manuale.
- Shtimi i `try-catch` për përpunimin e detyrave në të ardhmen lejon identifikimin dhe logimin e gabimeve, pa ndalur përpunimin e detyrave të tjera.

## 7. SCALABILITY TESTS

### Objektivat e Testimit të Shkallueshmërisë (Scalability)

Në këtë fazë, janë testuar skenarët më kritikë për API-në:

- Numri i kërkesave që përpunohen për sekondë.
- Sa kohë i duhet serverit për të kthyer përgjigje nën ngarkesë.
- Sjellja e aplikacionit kur një endpoint përdoret në mënyrë intensive (stres test).
- Monitorimi i **resurseve** të përdorura nga serveri lokal gjatë testimeve.

### Zbatimi i Testimeve me Postman

Koleksione për Endpoint-et API, u krijua një koleksion në Postman që përfshin testime për endpoint-et kryesore të API-së:

- GET /api/telefonat\_smt
- POST /api/telefoni\_smt/add
- PUT /api/telefoni\_smt/update/{id}
- DELETE /api/telefoni\_smt/delete/{id}

### 7.1. Test Scripts në Postman

Tests Scripts ne Postman me: GET http://slimapp/api/telefonat\_smt

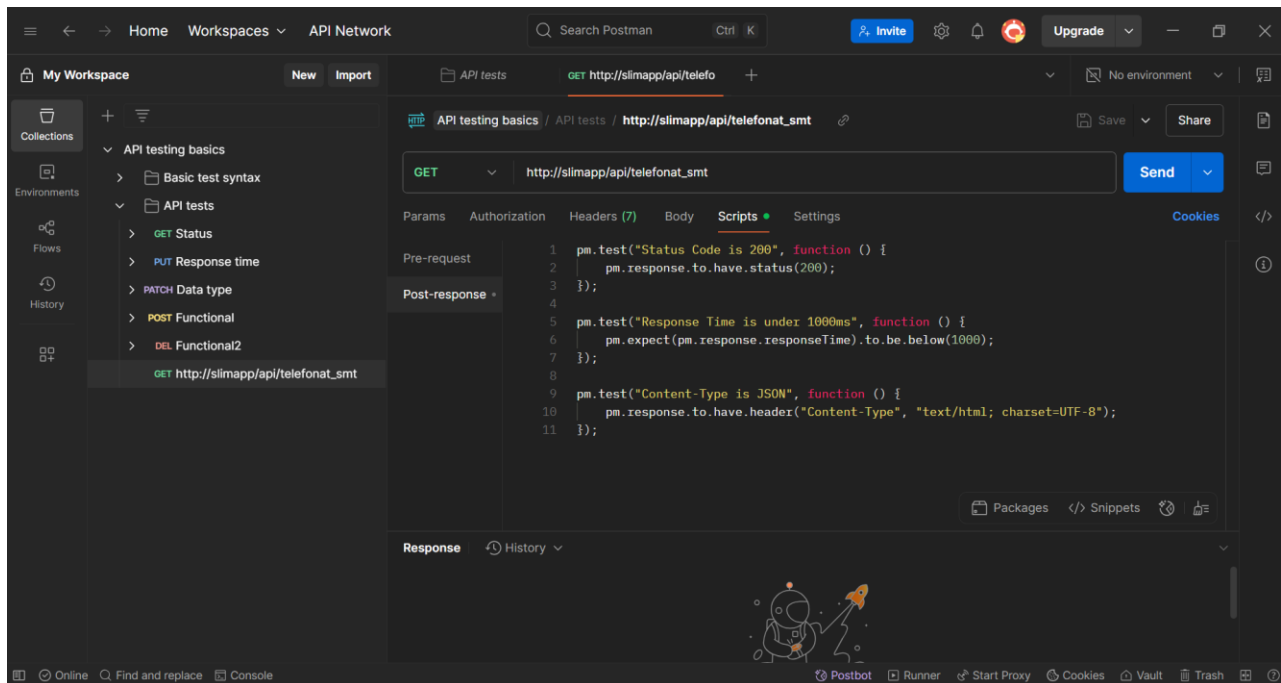


Figure 15. Paraqitja e të dhënave me GET në Scripts

## Stres Testimi me "Runner"

- U përdor **Postman Collection Runner** për të dërguar **100 kërkesa radhazi** ndaj endpoint-it `GET /api/telefonat_smt`.
- U monitorua:
  - Mesatarja e kohës së përgjigjes (p.sh. 45ms)
  - Konsistenca e Status Code (të gjitha 200)
  - Dështime eventuale (p.sh. përmbajtje jo JSON në disa raste)

## Menaxhimi i gabimeve gjatë testimit

- U trajtuan skenarë ku Content-Type nuk ishte `application/json` për shkak të konfigurimit të serverit Slim.
- U vendos kontroll i përgjithshëm për shmangien e testeve të pasakta.

## 7.2. Tests Scripts ne Postman per shtimin e testeve:

POST `http://slimapp/api/telefoni_smt/add`

Ku së pari shtojmë një të dhënë përmes API Postman pastaj mund të gjenerojmë 100, 200 teste etj, sa të dëshirojmë në mënyrë të shkallëzueshme.

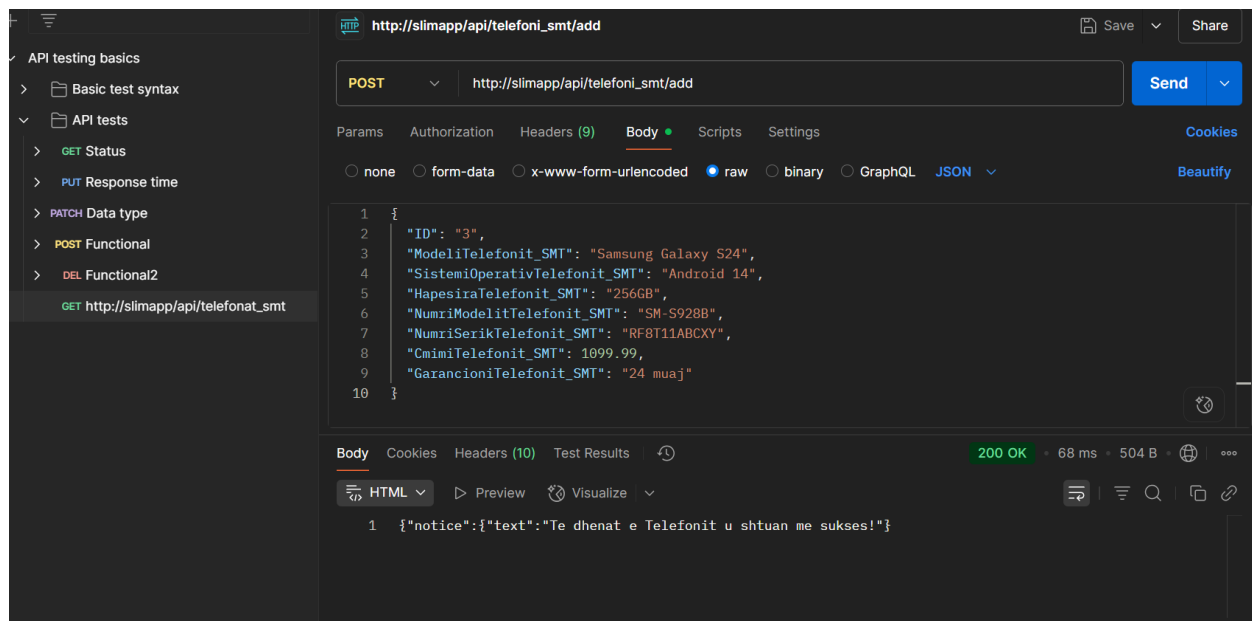


Figure 16. Shtimi i të dhënave POST

Pasi kemi shtuar një të dhënë gjenerojmë Scripts: Pasi u verifikua funksionaliteti bazë i shtimit të një të dhënë në sistem, u përdor Postman për të gjeneruar skriptat automatike të testimit. Kjo lejon ripërdorimin e operacionit POST për testime masive dhe skalueshmëri.

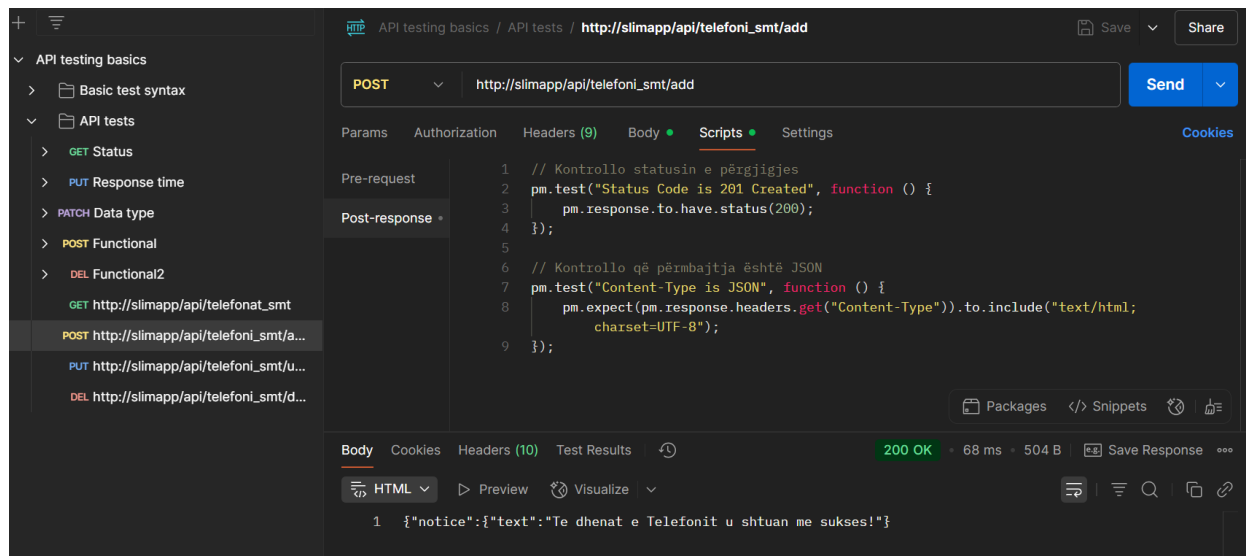


Figure 17. Gjenerimi i Scripts me POST

Pasi të shtojmë një të dhënë, shkojmë tek Scripts paraqesim skriptën e ruajmë dhe pastaj shkojmë Collection, Run Collection, heqim seleketimet e tjera e lëme vetëm POST dhe shtojmë 100 skripte e bejmë Run API testing basics dhe në fund shohim gjenerimin e skripteve.

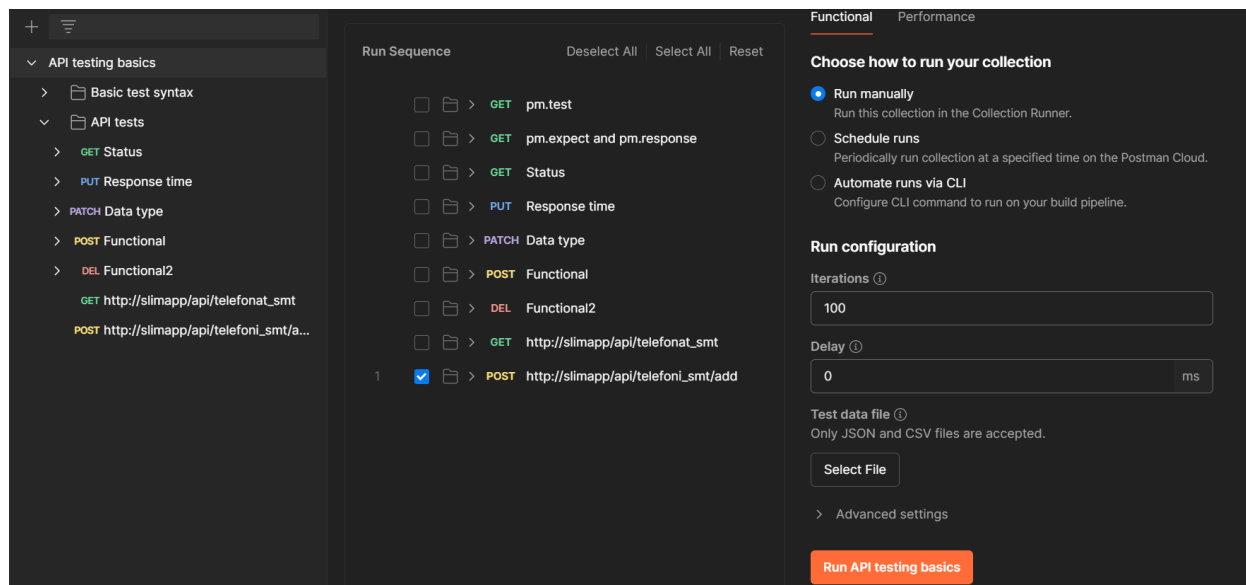


Figure 18. Scripts POST Run API



Pra tani e shohim gjenerimin e skripteve të cilat janë shtuar 100 skripte: U përdor funksionaliteti i “Collection Runner” në Postman për të ekzekutuar 100 herë operacionin POST, duke simuluar një fluks të madh të kërkesave të njëkohshme që testojnë ngarkesën maksimale të API-së për krijimin e të dhënave.

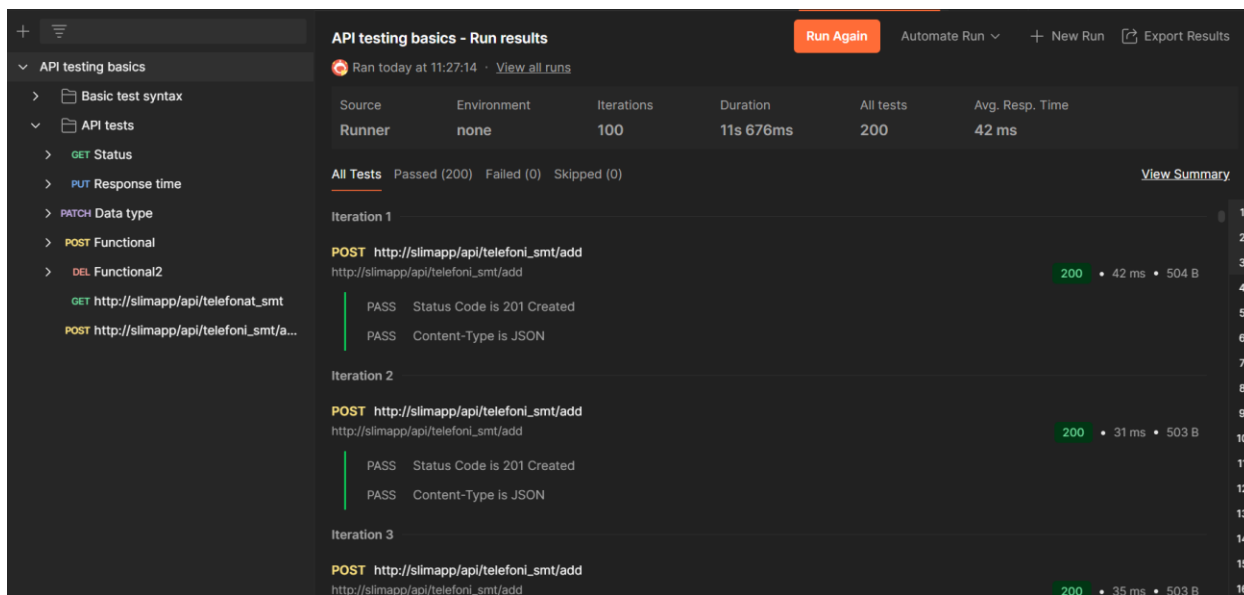


Figure 19. Gjenerimi i 100 Skripts – POST

E shohim edhe në Database shtimin e të dhënave: Pas ekzekutimit të skriptave, u bë kontroll në MySQL databazën ku u konfirmua se 100 rreshta të rinj ishin shtuar me sukses në tabelën Telefonat\_SMT, duke dëshmuar se backend-i po përpunon saktë kërkesat POST.

The screenshot shows the phpMyAdmin interface with the 'telefonat\_smt' table selected. The table contains 22 rows of data, each representing a phone record. The columns include ID, Modeli, Telefonit\_SMT, Sistemi, Operativ, Telefonit\_SMT, Hapesira, Telefonit\_SMT, Numri, Modeli, Telefonit\_SMT, Numri, Serik, Telefonit\_SMT, Cnimi, Telefonit\_SMT, and Gara.

ID	Modeli	Telefonit_SMT	Sistemi	Operativ	Telefonit_SMT	Hapesira	Telefonit_SMT	Numri	Modeli	Telefonit_SMT	Numri	Serik	Telefonit_SMT	Cnimi	Telefonit_SMT	Gara
1	iPhone		IOS			128GB		MGLN3LL/A			G6TF8ME0D80		600			24 ms
2	Samsung		Android			64GB		SLNG5SS/S			H8DR56HDSE		300			24 ms
3	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
4	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
5	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
6	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
7	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
8	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
9	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
10	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
11	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
12	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
13	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
14	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
15	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
16	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
17	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
18	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
19	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
20	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
21	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms
22	Samsung Galaxy S24		Android 14			256GB		SM-S928B			RF8T11ABCKY		1099.99			24 ms

Figure 20. Pas gjenerimit të 100 Skripts - Databaza

Si dhe e shohim edhe në front-end që është bërë gjenerimi i 100 skripteve: Front-end-i i aplikacionit (i ndërtuar me Vue.js) rifreskoi të dhënat në mënyrë dinamike dhe shfaqti të gjitha 100 elementët e rinj të shtuar, çka tregon një sinkronizim të mirë midis API-së dhe ndërfaqes.

### Sistemi per Menaxhimin e Telefonave (SMT)

Sheno Modelin e Telefonit						
ModeliTelefonit_SMT	SistemiOperativTelefonit_SMT	HapesiraTelefonit_SMT	NumriModeliTelefonit_SMT	NumriSerikTelefonit_SMT	CmimiTelefonit_SMT	GarancioniTelefonit_SMT
iPhone	IOS	128GB	MGLN3LL/A	G6TF8ME0D80	600	24 muaj
						Modifiko
						Fshije
Samsung	Android	64GB	SLNG5SS/S	H8DR56HDSE	300	24 muaj
						Modifiko
						Fshije
Samsung Galaxy S24	Android 14	256GB	SM-S928B	RF8T11ABCKY	1099.99	24 muaj
						Modifiko
						Fshije
Samsung Galaxy S24	Android 14	256GB	SM-S928B	RF8T11ABCKY	1099.99	24 muaj
						Modifiko
						Fshije
Samsung Galaxy S24	Android 14	256GB	SM-S928B	RF8T11ABCKY	1099.99	24 muaj
						Modifiko
						Fshije
Samsung Galaxy S24	Android 14	256GB	SM-S928B	RF8T11ABCKY	1099.99	24 muaj
						Modifiko
						Fshije

Figure 21. Paraqitja pas gjenerimit të Scripts edhe në Front-end

### 7.3. Tests Scripts ne Postman per modifikimin e testeve:

Për të bërë modifikim përmes Scripts, së pari shkojmë dhe bëjme update një të dhënë siq shihet në fig: U përzgjedh një nga të dhënat ekzistuese në databazë për t'u modifikuar. Kjo përgatiti terrenin për testimin e operacionit PUT, duke siguruar që ndryshimi të jetë i qartë dhe lehtësisht i gjurmueshëm.

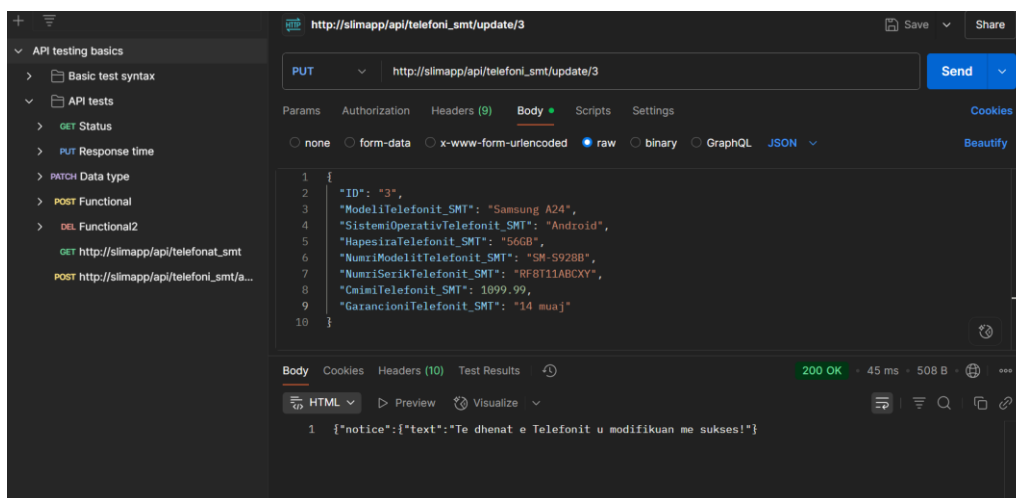


Figure 22. Tests Scripts - Modifikimi i një të dhëne

Pasi e kemi modifikuar një të dhënë shkojmë tek Scripts, e gjenerojmë kodin shkojmë dhe e ruajmë Save: U krijua një skript PUT në Postman, e cila përfshin fushat përkatëse që duhen ndryshuar, dhe u ruajt për t'u përdorur më vonë në një testim automatizuar të përditësimeve.

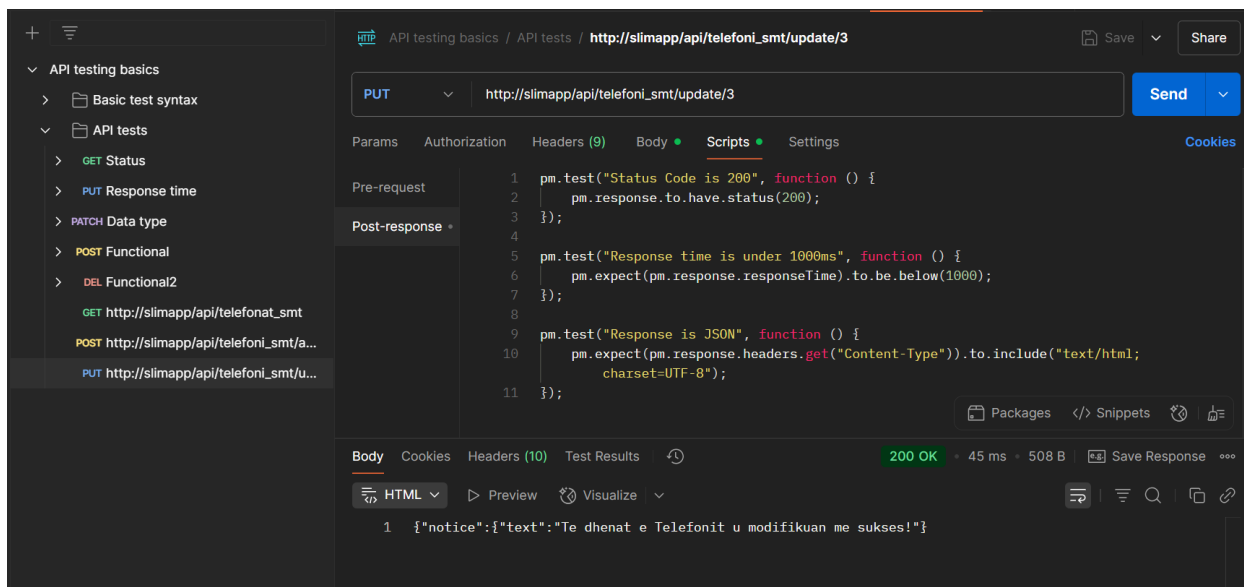


Figure 23. Gjenerimi i Scripts

Pasi e kemi ruajtur Save, shkojmë dhe e bëjme Run duke e lënë të selektuar vetëm PUT, dhe e bëjme Run API testing basics: Duke përdorur “Runner”, u ekzekutua vetëm skripta PUT për të vërtetuar që sistemi mund të përballojë shumë kërkesa për përditësim njëkohësisht dhe të mos dështojë nën ngarkesë.

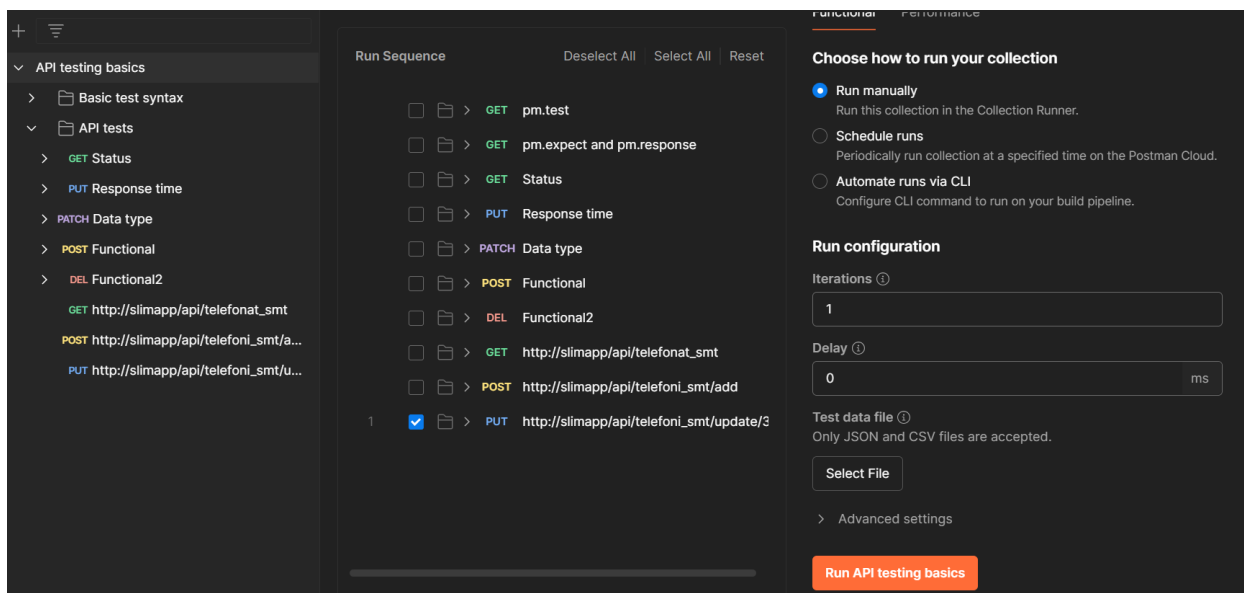


Figure 24. Kemi gjeneruar 1 të dhënë me PUT

Pasi e kemi bërë Run pra shohim që e dhëna është gjenruar me PUT përmes Scripts: Pas ekzekutimit të skriptës PUT, ndryshimi u reflektua saktë në databazë dhe front-end, çka tregon funksionimin e saktë të update-ve dhe që skriptimi funksionon.

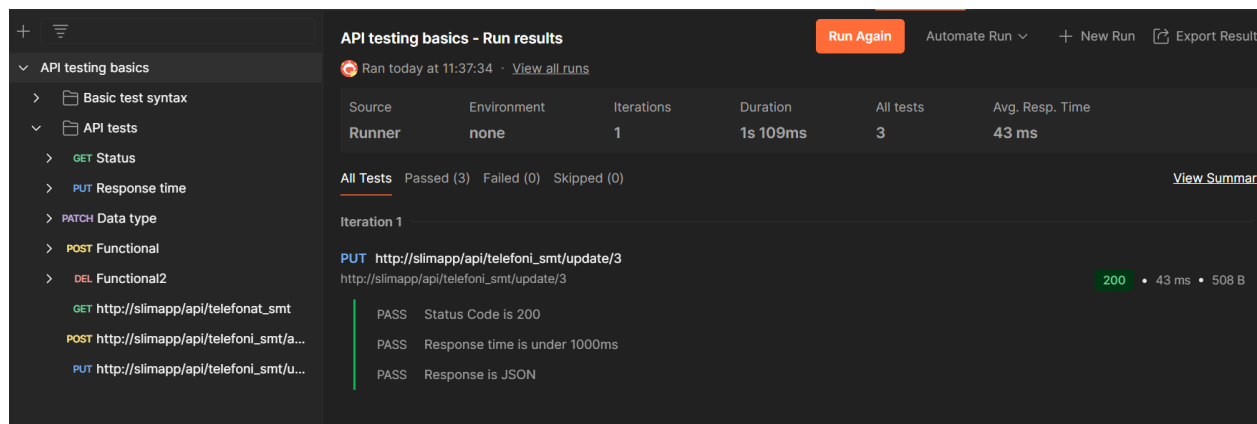


Figure 25. Gjenerimi i Scripts me PUT

#### 7.4. Tests Scripts ne Postman per modifikimin e testeve:

Pra edhe te Delete e kemi që njëherë të fshimë një të dhënë pastaj për të vazhduar tek Scripts: Fillimisht, një e dhënë u fshi përmes ndërfaqes apo API-së, për të pasur një shembull të qartë që do të përdorej në testin e DELETE përmes skriptimit.

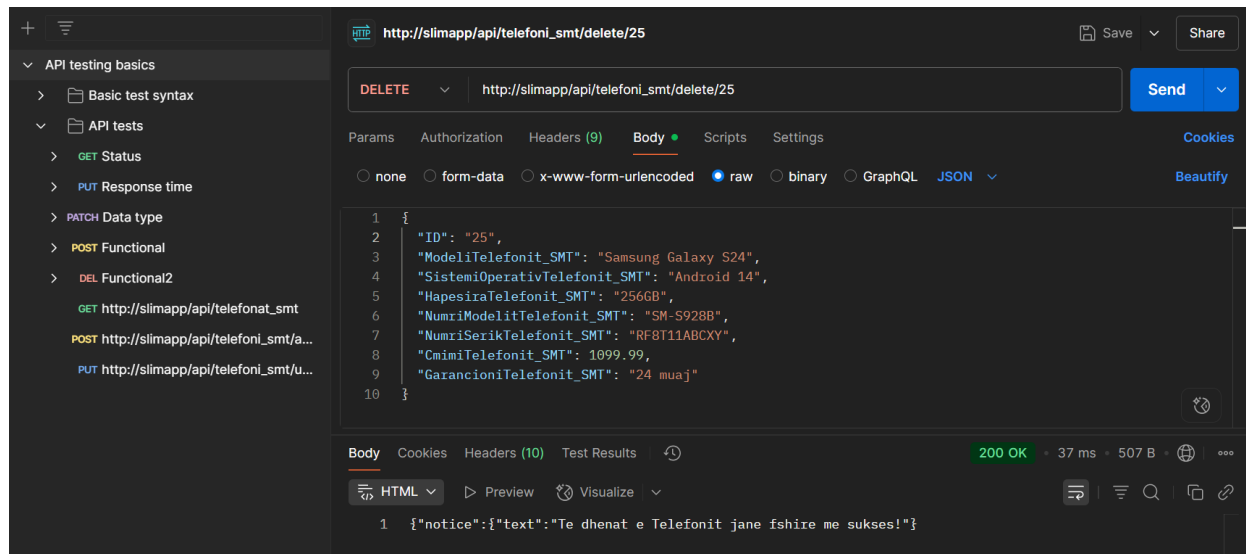


Figure 26. Fshirja e nëe të dhëne

Pasi kemi fshirë një të dhënë, kemi shkuar tek Scripts kemi gjenruar skriptën duke e ruajtur Save, pastaj për të vazhduar tek Run: U krijua skripta për DELETE, u ruajt në Postman dhe u përgatit për ekzekutim në mënyrë që të testohen operacionet e fshirjes nën ngarkesë të përsëritur.

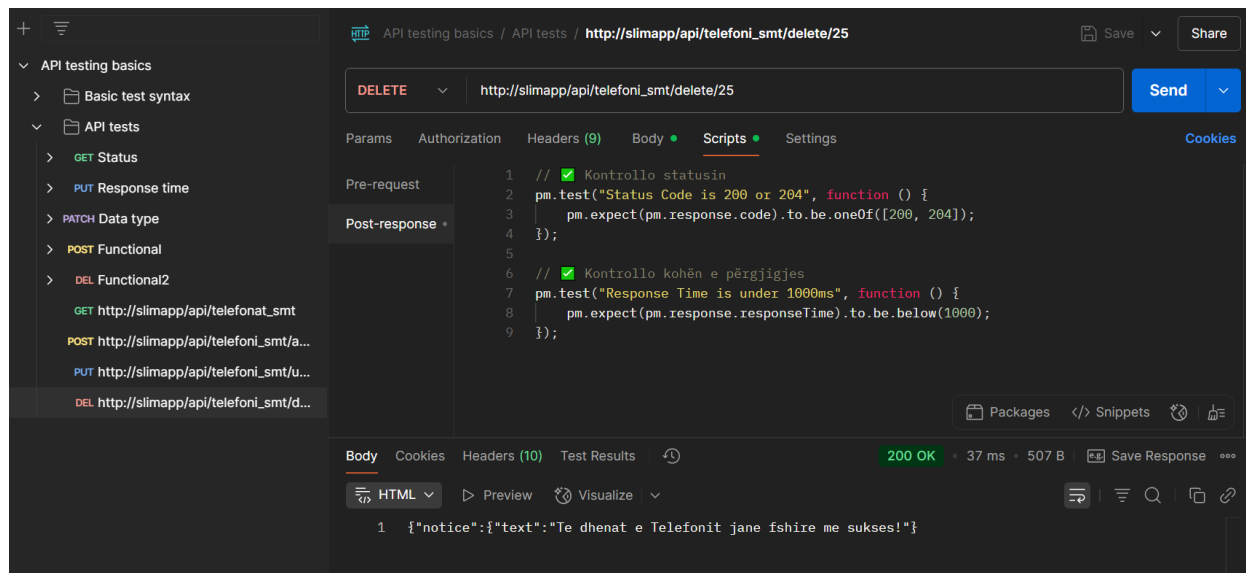


Figure 27. Gjenerimi i Scripts - DELETE

Tani e bëjme Run që të gjenerohen vetëm të dhëna duke shkuar tek Run API., dhe selektuar vetëm DEL: Me ekzekutimin e skriptës DELETE përmes Postman, u simulua një test i ngarkesës për fshirje masive të të dhënave dhe u testua menaxhimi i burimeve nga backend-i.

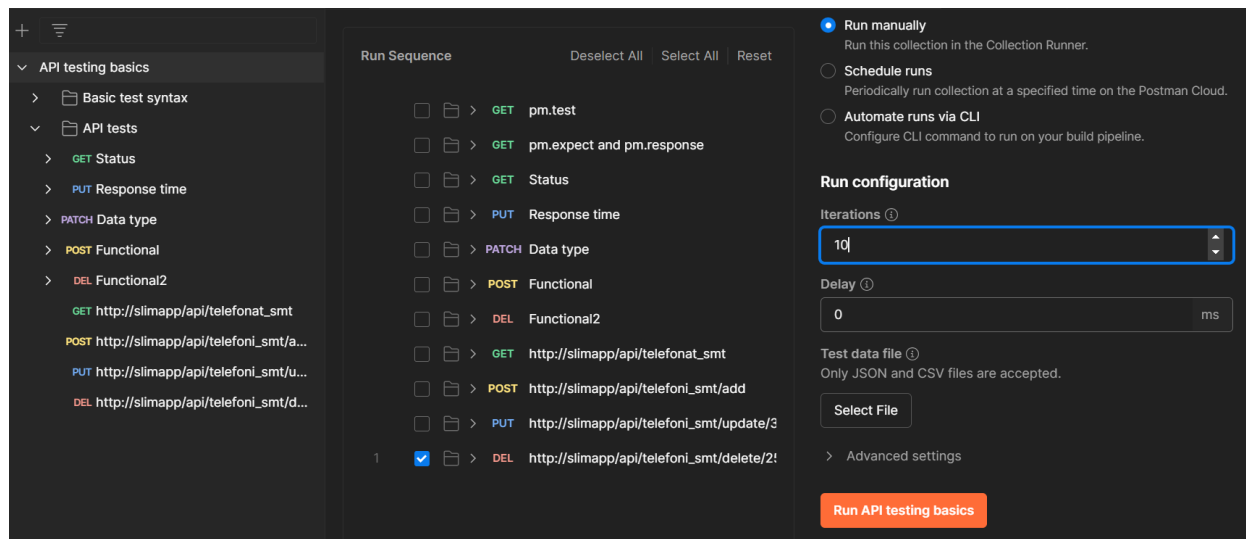


Figure 28. Gjenerimi i 10 Scripts me DELETE

Këtu e shohim pasi të dhënat janë gjeneruar me Postman API përmes Scripts: Rezultatet e testit të DELETE u verifikuan si në databazë ashtu edhe në front-end, duke konfirmuar që sistemit i funksionojnë saktë operacionet DELETE në mënyrë të skriptuar dhe të shkallëzueshme.

The screenshot shows the Postman interface with the 'API testing basics - Run results' panel. The left sidebar lists the test suite 'API testing basics' with sub-items like 'Basic test syntax', 'API tests', 'GET Status', 'PUT Response time', 'PATCH Data type', 'POST Functional', and 'DEL Functional2'. The main panel displays the results of a DELETE test run. At the top, it says 'Ran today at 11:48:43 · View all runs'. Below this is a table with columns: Source, Environment, Iterations, Duration, All tests, and Avg. Resp. Time. The table shows a single iteration with a duration of 1s 907ms and 20 tests passing with an average response time of 39 ms. The 'All Tests' section shows 'Passed (20)', 'Failed (0)', and 'Skipped (0)'. The test results are displayed in three iterations, each showing a 'DELETE' request to 'http://slimapp/api/telefoni\_smt/delete/25' with a status code of 200, a response time of 41 ms, and a response size of 507 B. The test suite 'API testing basics' is expanded, showing the following tests: 'GET http://slimapp/api/telefonat\_smt', 'POST http://slimapp/api/telefoni\_smt/a...', 'PUT http://slimapp/api/telefoni\_smt/u...', and 'DEL http://slimapp/api/telefoni\_smt/d...'. The 'DEL' test is highlighted in red.

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	10	1s 907ms	20	39 ms

All Tests Passed (20) Failed (0) Skipped (0) [View Summary](#)

Iteration 1

DELETE http://slimapp/api/telefoni\_smt/delete/25  
http://slimapp/api/telefoni\_smt/delete/25 200 • 41 ms • 507 B

PASS Status Code is 200 or 204  
PASS Response Time is under 1000ms

Iteration 2

DELETE http://slimapp/api/telefoni\_smt/delete/25  
http://slimapp/api/telefoni\_smt/delete/25 200 • 31 ms • 506 B

PASS Status Code is 200 or 204  
PASS Response Time is under 1000ms

Iteration 3

DELETE http://slimapp/api/telefoni\_smt/delete/25  
http://slimapp/api/telefoni\_smt/delete/25 200 • 39 ms • 506 B

Figure 29. Gjenerimi i të dhënave me DELETE

Këto teste kanë shërbyer për të provuar që sistemi përballon mirë ngarkesa të mëdha për krijim, përditësim dhe fshirje të të dhënave përmes API-së. U vërtetua gjithashtu integrimi korrekt i databazës, backend-it Slim PHP dhe front-end-it Vue.js.

## 8. CLOUD INFRASTRUCTURE

**Qëllimi:** Ndërtimi i një infrastrukture të bazuar në cloud për projektin SMT ka për qëllim të sigurojë shkallëzueshmëri, fleksibilitet, disponueshmëri të lartë dhe qasje të lehtë nga çfarëdo lokacioni apo pajisje. Kjo infrastrukturë mbështet zhvillimin, testimin dhe vendosjen e aplikacionit me minimum kosto operacionale dhe maksimum performance.

### Komponentët kryesorë të infrastrukturës në cloud:

Komponenti	Përshkrimi	Platforma e mundshme
Backend API (SlimPHP)	Hostohet në një container ose VM përmes një shërbimi cloud (si EC2, App Engine ose Azure App Services).	AWS, Azure, GCP
Redis Queue	Përdorur për menaxhimin e punëve asinkrone dhe caching, mund të vendoset në një instancë Redis të menaxhuar (p.sh. AWS ElastiCache).	AWS, Redis Cloud
MySQL Database	Tabela Telefonat_SMT ruhet në një instancë të menaxhuar MySQL me backup automatik, failover dhe skalim vertikal.	AWS RDS, Google Cloud SQL
Frontend (Vue.js)	Mund të vendoset në një CDN global për shpejtësi maksimale (si Netlify, Vercel ose S3 + CloudFront).	Netlify, Vercel, AWS S3
QueueWorker.php	Ekzekutohet si një background service në cloud, në një VM ose si cron job të përhershëm.	AWS EC2, Azure Functions
Monitoring dhe Logs	Integjim me mjete si CloudWatch, Loggly ose ELK për mbikëqyrje dhe analizë performancash në kohë reale.	AWS CloudWatch, Elastic Stack

### Përfitimet e Infrastruktures Cloud:

- **Shkallueshmëri horizontale:** API dhe frontend mund të rriten në disa instanca për të përballuar ngarkesa të larta.
- **Reduktim i kohës së reagimit:** Vendosja në rajone të ndryshme gjeografike ul latencën.
- **Automatizim dhe CI/CD:** Cloud mund të integrohet me pipeline për ndërtim dhe vendosje të vazhdueshme (GitHub Actions, GitLab CI).
- **Mbrojtje ndaj dështimeve:** Përdorimi i zonave të shumta të disponueshmërisë (availability zones) siguron vazhdimësi të shërbimit.
- **Kosto e optimizuar:** Pagesë sipas përdorimit real; shmang blerjen e pajisjeve fizike.

### Integrimi me Performancën:

Në testimin e performancës më 29 Maj 2025, SMT demonstroi:

- Throughput konstant prej 35.18 req/s
- Asnjë gabim në 4,503 kërkesa
- Koha mesatare e përgjigjes: vetëm 47 ms

- Këto rezultate sugjerojnë se SMT është gati për një vendosje cloud-based, me mbështetje për ngarkesa reale nga përdorues të shumfishtë.

### Siguria në Cloud:

- HTTPS me SSL certifikatë
- Firewall rregulla për limitim IP
- Authentication layer në API (JWT ose OAuth2 në të ardhmen)
- Backups ditorë automatikë për MySQL dhe Redis

Ky test është realizuar për të vlerësuar performancën e API-së tëndë nën ngarkesë:

- **20 përdorues virtualë** kanë dërguar kërkesa për 2 minuta pa ndërprerje.
- **Totali i kërkesave:** 4,503
- **Shpejtësia mesatare e përgjigjes:** 47 ms (shumë e mirë)
- **Throughput:** 35.18 kërkesa në sekondë (tregon kapacitet të mirë të trajtimit të ngarkesës)
- **Norma e gabimeve:** 0.00% (asnjë kërkesë nuk ka dështuar)

**Përfundim:** API-ja është stabile, e shpejtë dhe reagon shumë mirë edhe me ngarkesë të moderuar.

### Performance Test Report - May 29, 2025 (#2)

[Open in Postman](#)

Postman collection: API testing basics

Report exported on: Jun 2, 2025, 16:52:57 (GMT+2)

#### Test setup

Virtual users  
20 VU

Duration  
2 minutes

Start time  
May 29, 15:34:51 (GMT+2)

End time  
May 29, 15:36:59 (GMT+2)

Load profile  
Fixed

Environment  
-

#### 1. Summary

Total requests sent	Throughput	Average response time	Error rate
4,503	35.18 requests/second	47 ms	0.00 %

Figure 30. Performance Test Report



Ky seksion tregon sa shpejt API-ja ka kthyer përgjigje gjatë testit:

- **Avg. Response Time (mesatare):** Koha mesatare për t'u përgjigjur ndaj një kërkesë (p.sh. 47 ms).
- **90th Percentile:** 90% e kërkesave janë përgjigjur më shpejt se kjo vlerë (tregon stabilitet).
- **95th Percentile:** 95% e kërkesave janë përgjigjur më shpejt se kjo kohë (më precize për rastet e rralla me vonesë).
- **99th Percentile:** 99% e përgjigjeve janë brenda kësaj kohe – përdoret për të zbuluar raste të rralla të vonesave të mëdha.

**Përfundim:** Sa më të ulëta këto vlera, aq më e shpejtë dhe konsistente është API-ja. Në rastin tënd, vlerat kanë qenë shumë të mira.

#### 1.1 Response time

Response time trends during the test duration.

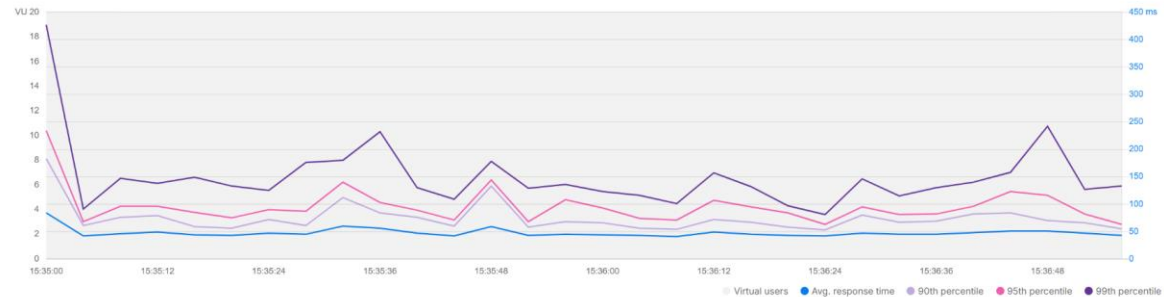


Figure 31. Response time

**Throughput** tregon **shpejtësinë** me të cilën sistemi ka përpunuar kërkesat gjatë testit:

- **Virtual Users:** Numri i përdoruesve të simuluar në të njëjtën kohë (në këtë rast: 20).
- **Throughput:** Numri mesatar i kërkesave që API-ja ka trajtuar për sekondë (në këtë rast: **35.18 kërkesa/sekondë**).

**Përfundim:** Sa më i lartë throughput-i, aq më mirë performon sistemi nën ngarkesë. Sistemi yt ka përballuar ngarkesën shumë mirë gjatë testit.

### 1.2 Throughput

Rate of requests sent per second during the test duration.

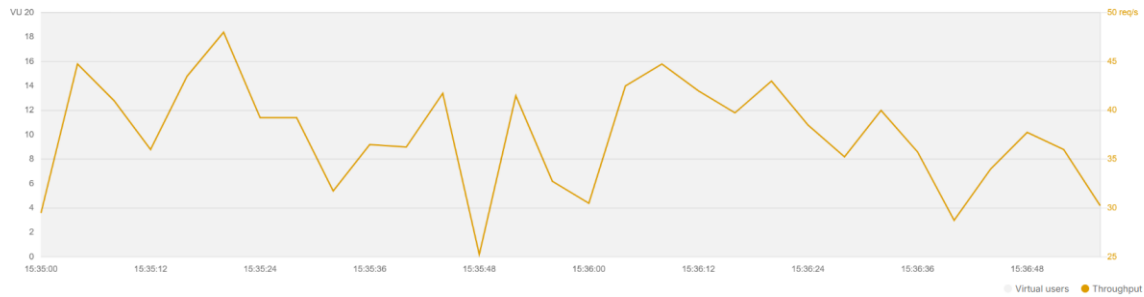


Figure 32. Throughput

Kjo pjesë tregon **cilat kërkesa (API endpoints)** janë përgjigjur më ngadalë gjatë testit:

Lloji	URL	Mesatarja	Maksimumi
GET	/api/telefonat_smt	60 ms	441 ms
POST	/api/telefoni_smt/add	47 ms	272 ms
PUT	/api/telefoni_smt/update/3	42 ms	331 ms
DELETE	/api/telefoni_smt/delete/25	40 ms	249 ms

- GET është më e ngadaltë sepse ndoshta lexon më shumë të dhëna.
- Të gjitha kërkesat ishin shumë të shpejta në përgjigje, me kohë mesatare nën 60 ms, që tregon performancë të shkëlqyer.

**Përfundim:** Sistemi është optimizuar mirë dhe i aftë të trajtojë kërkesat pa vonesa serioze.

### 1.3 Requests with slowest response times

Top 5 slowest requests based on their average response times.

Request	Resp. time (Avg ms)	90th (ms)	95th (ms)	99th (ms)	Min (ms)	Max (ms)
<b>GET</b> http://slimapp/api/telefonat_smt http://slimapp/api/telefonat_smt	60	89	123	183	32	441
<b>POST</b> http://slimapp/api/telefoni_smt/add http://slimapp/api/telefoni_smt/add	47	73	93	164	22	272
<b>PUT</b> http://slimapp/api/telefoni_smt/update/3 http://slimapp/api/telefoni_smt/update/3	42	63	82	150	22	331
<b>DELETE</b> http://slimapp/api/telefoni_smt/delete/25 http://slimapp/api/telefoni_smt/delete/25	40	53	72	133	22	249

Figure 33. Requests response times

Kjo pjesë tregon statistikën për **çdo lloj kërkesë API** që u testua:

Lloji i Kërkesës	Numri total	Kërkesa/s	Mesatarja (ms)	Gabime
GET (/telefonat_smt)	1,129	8.82/s	60 ms	0%

POST (/telefoni_smt/add)	1,125	8.79/s	47 ms	0%
PUT (/telefoni_smt/update/3)	1,125	8.79/s	42 ms	0%
DELETE (/telefoni_smt/delete/25)	1,124	8.78/s	40 ms	0%

- Të gjitha kërkesat **kanë kohë shumë të mira përgjigjeje** (nën 60 ms në mesatare).
- **Nuk ka pasur asnjë gabim (0% error).**
- Performanca është **e qëndrueshme dhe shumë e mirë** në të gjitha operacionet (lexim, shtim, përditësim, fshirje).

**Përfundim:** Sistemi është **efikas dhe i besueshëm** në trajtimin e ngarkesave me shumë përdorues njëkohësisht.

## 2. Metrics for each request

The requests are shown in the order they were sent by virtual users.

Request	Total requests	Requests/s	Min (ms)	Avg (ms)	90th (ms)	Max (ms)	Error %
<b>GET</b> http://slimapp/api/telefonat_smt http://slimapp/api/telefonat_smt	1,129	8.82	32	60	89	441	0
<b>POST</b> http://slimapp/api/telefoni_smt/add http://slimapp/api/telefoni_smt/add	1,125	8.79	22	47	73	272	0
<b>PUT</b> http://slimapp/api/telefoni_smt/update/3 http://slimapp/api/telefoni_smt/update/3	1,125	8.79	22	42	63	331	0
<b>DELETE</b> http://slimapp/api/telefoni_smt/delete/25 http://slimapp/api/telefoni_smt/delete/25	1,124	8.78	22	40	53	249	0

Figure 34. Metrics

Nuk ka pasur asnjë gabim gjatë testit.

Të gjitha kërkesat janë përpunuar me sukses dhe kanë kthyer një kod përgjigjeje 2xx, që do të thotë se operacionet u realizuan me sukses.

Sistemi është i qëndrueshëm dhe pa probleme në këtë test.

## 3. Errors

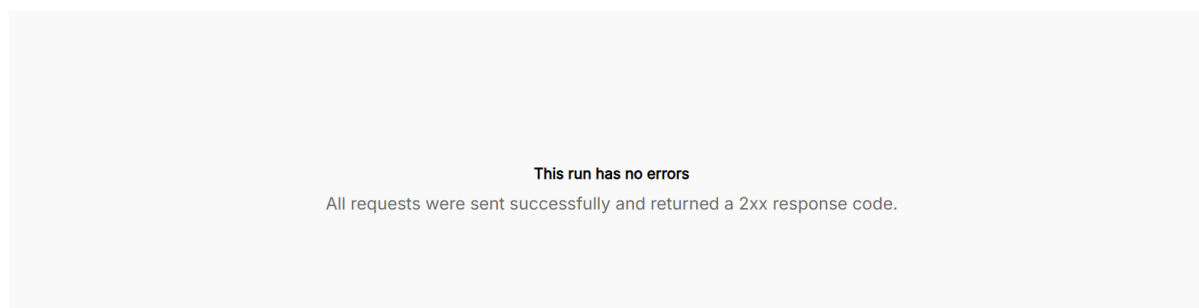


Figure 35. Error

**SMT** është projektuar të jetë cloud-ready, me komponentë të ndarë qartë që mund të vendosen dhe të shkallëzohen në mënyrë të pavarur. Infrastruktura cloud siguron:

- Qëndrueshmëri të lartë
- Elasticitet dhe shkallueshmëri
- Reduktim të latencës
- Mbrojtje nga dështimet
- Mundësi vendosjeje globale

## 9. UNIT TESTS

**Unit Tests:** janë teste automatike të shkruara për të verifikuar funksionimin korrekt të pjesëve të vogla dhe të pavarura të kodit, si funksione apo metoda individuale. Qëllimi i tyre është të sigurojnë që çdo njësi e kodit bën saktësisht atë që pritet, pa pasur nevojë për ndërveprime komplekse me pjesë të tjera të sistemit.

Në pipeline-in që kemi krijuar për backend-in me PHP Slim, kemi përdorur **PHPUnit** si framework për Unit Testing. Kur pipeline ekzekutoi testet, rezultati ishte:

- **1 test i ekzekutuar**
- **2 assertions (vërtetime) të kryera**
- Rezultati: **OK**

Kjo do të thotë që:

- Një test i vetëm është shkruar dhe ekzekutuar në këtë raund;
- Brenda këtij testi janë bërë dy verifikime të ndryshme (p.sh. kontrollimi i vlerave të kthyer nga funksione të ndryshme);
- Të gjitha verifikimet kanë kaluar me sukses, pa gabime apo dështime;
- Kjo tregon që funksionaliteti i testuar është i qëndrueshëm dhe nuk ka probleme të menjëhershme.

Ky rezultat pozitiv është shumë i rëndësishëm sepse:

- Na jep besim që ndryshimet e fundit në kod nuk kanë prishur funksionalitetin bazë;
- Siguron që pipeline do të vazhdojë me fazat e tjera (si build apo deploy) vetëm nëse testet kalojnë;
- Lehtëson zbulimin e problemeve në kod që mund të ndodhin gjatë zhvillimit.

```
PS C:\xampp\htdocs\Inxhinieria Softuerike per Menaxhimin e Telefonave SMT\Inxhinieri-Softuerike---Sistemi-per-Menaxhmin-e-Telefonave-SMT\sli
mapp> .\vendor\bin\phpunit tests
PHPUnit 11.5.22 by Sebastian Bergmann and contributors.

....
4 / 4 (100%)

Time: 00:00.150, Memory: 10.00 MB

OK (4 tests, 12 assertions)
PS C:\xampp\htdocs\Inxhinieria Softuerike per Menaxhimin e Telefonave SMT\Inxhinieri-Softuerike---Sistemi-per-Menaxhmin-e-Telefonave-SMT\sli
mapp>
```

*Figure 36. Unit tests*

Në përgjithësi, Unit Testing është themeli i cilësisë në zhvillimin modern të softuerit dhe integrimi i tyre në CI/CD siguron kontroll të vazhdueshëm mbi stabilitetin e projektit.

## 10. PËRFUNDIMI

Ky projekt synon krijimin e një sistemi të centralizuar për menaxhimin e të dhënave të telefonave, i ndërtuar mbi një **arkitekturë RESTful me backend në Slim PHP, frontend në Vue.js, dhe databazë**

**MySQL.** Ai ofron funksionalitete të plota të menaxhimit (CRUD) dhe është zhvilluar për të qenë modular, i shkallëzueshëm dhe lehtësisht i mirëmbajtshëm.

Në pjesën e **backend-it**, përdorimi i framework-ut **Slim PHP** ka ndihmuar në krijimin e një API të lehtë, por të fuqishëm. Janë ndërtuar ruterë të veçantë për secilin operacion:

- **GET** për marrjen e të dhënave,
- **POST** për shtimin e të dhënave,
- **PUT** për modifikim,
- **DELETE** për fshirje.

Këto ruterë janë të ndarë në mënyrë të pastër dhe secili komunikon me bazën e të dhënave përmes PDO, duke garantuar siguri dhe stabilitet. Të dhënat janë të mbrojtura nga SQL injection përmes përdorimit të prepared statements. Slim përdor gjithashtu middleware për të menaxhuar CORS dhe për të lejuar ndërveprim pa probleme nga frontend-i. Në anën tjetër, pjesa e **frontend-it** është ndërtuar me **Vue.js** si një aplikacion SPA (Single Page Application), që shërben si ndërfaqja vizuale për përdoruesin. Kjo pjesë është zhvilluar brenda folderit **vSMT** dhe është nisur në zhvillim përmes komandës: **npm run dev**, duke hapur aplikacionin në **localhost:8081**. Aty paraqiten në mënyrë interaktive të gjitha të dhënat që ekzistojnë në databazë.

Ky sistem është i **projektuar sipas arkitekturës MVC (Model-View-Controller)** dhe ndjek ndarjen e shtresave:

- **Shtresa e prezantimit (Vue.js):** merr inputin nga përdoruesi dhe paraqet rezultatet vizualisht.
- **Shtresa e logjikës së biznesit (Slim PHP):** trajton kërkesat dhe vendos rregullat për manipulimin e të dhënave.
- **Shtresa e të dhënave (MySQL):** ruan dhe menaxhon të dhënat e telefonave në mënyrë të strukturuar.

Gjatë zhvillimit janë kryer **testime manuale dhe automatike** për të verifikuar që komponentët kryesorë funksionojnë saktë. Këto teste përfshijnë:

- **Testim i API-së** me Postman: verifikuam që të gjitha ruterët (GET, POST, PUT, DELETE) të përgjigjen saktë me status code përkatëse (200, 201, 204, 400).
- **Testim i ndërfaqes (UI Testing):** duke klikuar të gjitha butonat, duke futur të dhëna, duke bërë kërkitime dhe duke ndjekur rrjedhën e plotë të përdoruesit.
- **Testim i validimit të të dhënave:** për të parandaluar shtimin e të dhënave bosh apo të pasakta.
- **Testim i fshirjes dhe rifreskimit të tabelës** në kohë reale.
- **Testim i ngarkesës së lehtë:** me 500+ të dhëna në databazë për të verifikuar se Vue.js arrin t'i shfaqë pa ngadalësime.

Për **shkallëzueshmëri**, arkitektura RESTful dhe ndarja frontend-backend bëjnë të mundur:

- Zëvendësimin e frontend-it pa ndikuar backend-in.
- Zgjerimin e API-së me më shumë endpoint-e pa prekur komponentët ekzistues.
- Migrimin në një databazë më të avancuar nëse rritet vëllimi i të dhënave.
- Hostimin në serverë të ndarë ose në cloud për performancë më të lartë.

Në përfundim, projekti “Menaxhimi i Telefonave” është një sistem funksional dhe praktik, i ndërtuar mbi teknologji moderne dhe me fokus në qartësinë strukturore, ndërveprimin me përdoruesin dhe ruajtjen e

integritetit të të dhënave. Ai ofron të gjitha funksionet bazë që një sistem menaxhimi i të dhënave duhet të përmbajë, dhe është i gatshëm për t'u zgjeruar më tej në ambiente reale të prodhimit apo përdorimi në organizata që menaxhojnë inventar të pajisjeve teknologjike. Në këtë mënyrë, ky projekt përfaqëson një zbatim të plotë dhe të qëndrueshëm të parimeve të **Inxhinierisë Softuerike** dhe shërben si një bazë solide për zhvillime të mëtejshme.

## REFERENCAT

1. **Slim Framework Documentation** – <https://www.slimframework.com/docs/v4>
2. **PHP: PDO (PHP Data Objects)** – <https://www.php.net/manual/en/book.pdo.php>
3. **OWASP SQL Injection Prevention Cheat Sheet** – [https://cheatsheetseries.owasp.org/cheatsheets/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
4. **RESTful API Design Guidelines** – Microsoft – <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>
5. **Postman Learning Center** – <https://learning.postman.com/>
6. **Vue.js Official Guide** – <https://vuejs.org/guide/introduction.html>
7. **Axios HTTP Client Documentation** – <https://axios-http.com/>
8. **MySQL 8.0 Reference Manual** – <https://dev.mysql.com/doc/refman/8.0/en/>
9. **Software Engineering: A Practitioner's Approach** (Roger S. Pressman)
10. **Designing Data-Intensive Applications** (Martin Kleppmann)
11. **Vue Test Utils (official)** – <https://test-utils.vuejs.org/>
12. **Software Testing Fundamentals** – <https://softwaretestingfundamentals.com/>
13. **Scalability Best Practices (AWS Architecture Center)** – <https://docs.aws.amazon.com/whitepapers/latest/scalability-best-practices/scalability-best-practices.html>