Универзитет у Крагујевцу

Информационе технологије



Домаћи задатак

Рачунарство у облаку

Тема:

**Провера испуњењости фактора 12-факторских апликација на апликацији за управљање библиотеком**

|  |  |  |
| --- | --- | --- |
| Студент: | | Предметни професор: |
| Вељко Јеленковић 2IT/2023 | проф. др Милош Ивановић | |
|  | |  |

Крагујевац, 2024.

**Садржај**

[**1.** **Увод у 12-факторске апликације** 3](#_Toc169654054)

[**2.** **Апликација за управљање библиотеком** 4](#_Toc169654055)

[**3.** **Да ли је апликација 12-факторна?** 6](#_Toc169654056)

# **Увод у 12-факторске апликације**

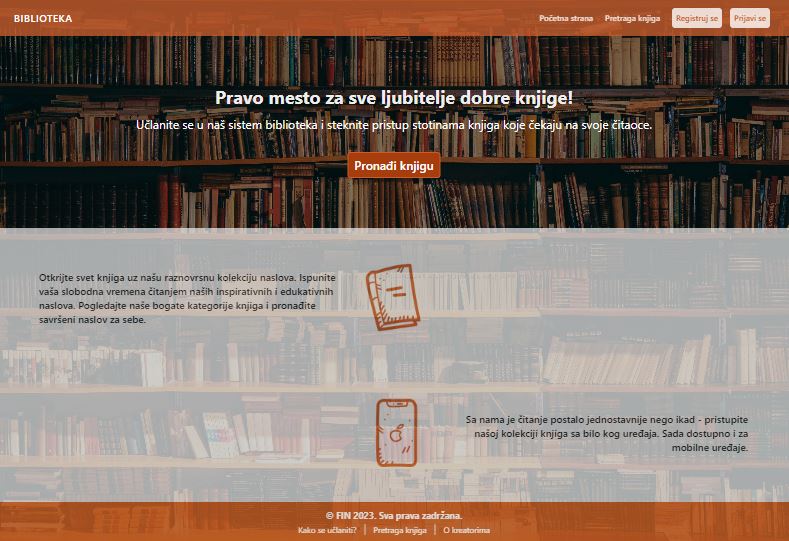
12-факторске апликације чине методологију за развој модерних апликација која помаже у изградњи апликација које су скалабилне, поуздане и лакше одрживе. Ова методологија је популаризована од стране тима у Хероку-у и заснива се на годинама искуства у развоју cloud-native апликација. Ево кључних принципа 12-факторских апликација:

1. **Код (Codebase):** Једна апликација би требало да има јединствени базни код са више deploy-а, чиме се омогућава лакше управљање променама и верзијама.
2. **Зависности (Dependencies):** Све зависности апликације треба јасно декларисати и изоловати, обично кроз конфигурационе фајлове као што су package.json за Node.js или pom.xml за Java пројекте.
3. **Конфигурација (Config):** Конфигурација апликације треба да буде смештена у променљивима окружења (environment variables), а не директно у коду. Ово омогућава флексибилност у различитим окружењима (развој, тестирање, продукција).
4. **Помоћни сервиси (Backing services):** Спољни сервиси као што су базе података, SMTP сервери или API-ји треба третирати као прикључне ресурсе који се могу лако заменити или скалирати независно од главне апликације.
5. **Изградња, издавање, покретање (Build, release, run):** Процесе изградње апликације (build), њеног објављивања (release) и покретања (run) треба строго раздвојити. Ово помаже у јасном дефинисању корака за deployment и смањењу могућности грешака.
6. **Процеси (Processes):** Апликација треба да буде дизајнирана као скуп одговорних, stateless процеса који могу лако да се скалирају хоризонтално и замене у случају неуспеха.
7. **Повезивање порта (Port binding):** Сервиси унутар апликације треба да буду експонирани преко повезивања портова. На пример, HTTP сервер би требало да слуша на одређеном порту који се може конфигурисати.
8. **Конкурентност (Concurrency):** Апликација треба да буде дизајнирана за скалирање путем конкурентних процеса, чиме се оптимизују ресурси и перформансе.
9. **Једноставна замена (Disposability):** Апликација треба брзо да стартује и да се гаси. Ово побољшава робусност апликације и омогућава лакше управљање у случају непланираних прекида.
10. **Паритет развоја и продукције (Dev/prod parity):** Окружења за развој, тестирање и продукцију треба да буду што сличнија како би се смањили проблеми који произилазе из разлика у конфигурацији и окружењу.
11. **Логови (Logs):** Апликација треба да генерише логове као токове догађаја који се могу лако анализирати и пратити, што помаже у откривању и решавању проблема.
12. **Административни процеси (Admin processes):** Административни задаци, као што су управљање базом података или извршавање миграција, треба да буду изоловани и извршавани као једнократни процеси.

Имплементација ових принципа помаже тимовима за развој апликација да побољшају скалабилност, робустност, брзину деплојмента и одрживост апликација у модерним cloud окружењима.

# **Апликација за управљање библиотеком**

Апликација за коју ће бити испитивани фактори јесте full-stack веб апликација, која има све неопходне алате и странице за управљање библиотеком и њеним огранцима, књигама и корисницима. Направљена је у MERN Stack-у и омогућава коришћење различитих улога (администратор и корисник), као и многе функције, укључујући претраживање и резервисање књига, читање детаљних описа и ПДФ-ова књига, обележавање омиљених књига и оних за праћење, додавање великог броја књига одједном, контролу резервација и многе друге. Такође укључује аутоматске е-поруке и СМС поруке као подсетнике, и мобилну апликацију за скенирање баркодова и лако додељивање књига.



*Слика 1. Апликација за управљање библиотеком*

У оквиру апликације, корисници могу да:

* приступе својој профилној страници и виде различите секције
* виде књиге које су тренутно њихове
* погледају тренутне резервације и историју резервација
* уређују своје профилне информације, као што су име, број телефона и лозинка
* виде препоручене књиге и оне које су обележили као омиљене
* приступе својим огранцима и њиховој локацији на мапи
* претражују, филтрирају претрагу и резервишу књиге
* виде детаљне информације о свакој књизи
* преузму ПДФ књиге, ако је доступан

Администратори могу да:

* виде све књиге и кориснике у свом огранку
* доделе књигу кориснику или је врате од корисника
* ажурирају или бришу било коју књигу у свом огранку
* додају листу књига у базу података преко Excel фајла
* додају једну књигу или једног аутора
* користе мобилну апликацију за скенирање баркодова и лако додељивање
* приступе детаљним информацијама о књизи преко мобилне апликације

Технологије и библиотеке коришћене за израду овакве апликације су:

* MERN Stack
* MongoDB - NoSQL база података
* Express.js - Node.js оквир за веб апликације
* React - JavaScript библиотека за изградњу корисничких интерфејса
* Node.js - JavaScript извршно окружење
* TailwindCSS - утилитарно-први CSS оквир
* Leaflet.js - JavaScript библиотека за интерактивне мапе
* React-toastify
* Mongoose - објектно моделирање за MongoDB у Node.js
* Xlsx – библиотека за рад са табелама
* Bcrypt - библиотека за хеширање лозинки
* jwt - библиотека за имплементацију JSON Web Tokens
* Nodemailer - Node.js модул за слање е-поште
* Gmail API
* Twilio - API за комуникацију за СМС, гласовне и видео поруке
* React Native - оквир за креирање мобилних апликација

Овај систем пружа свеобухватно решење за управљање библиотеком, од администрације и праћења књига до корисничких функција и мобилне подршке, што га чини идеалним за колеџе и образовне институције.

# **Да ли је апликација 12-факторска?**

У овом поглављу, осврнућемо се на 12 фактора која су раније поменута, и испитати их у контексту апликације за управљање библиотеком.

 **Код (Codebase)**:

* **Испуњено?** ❓
* Објашњење: Веб апликација има један базни код који се може deploy-овати на различита окружења. Међутим, мобилна апликација има свој засебан код, што је и уобичајено за апликације са засебним мобилним клијентом, али како не постоји дељени код између клијента мобилне и клијента веб апликације, не можемо сматрати да апликација има јединствени код. У зависности од тога како се гледа, уколико се мобилна апликација „не рачуна“, или уколико се уклони из репозиторијума, тек тада можемо сматрати код јединственим, обзиром да остаје само веб апликација која га примењује. Самим тим, сматра се да први фактор није испуњен.

 **Зависности (Dependencies)**:

* **Испуњено? ✅**
* Објашњење: Све зависности су јасно декларисане у конфигурационом фајлу package.json за Node.js пројекте. Што је још боље, постоје посебни package.json фајлови за сваки од три директоријума (client, mobile, server), што омогућава бољу модуларност и лакше лоцирање жељених зависности и библиотека. Можемо сматрати да је овај фактор испуњен.

 **Конфигурација (Config)**:

* **Испуњено? ✅**
* Објашњење: Конфигурација апликације је смештена у променљивама окружења, што омогућава флексибилност у различитим окружењима, обзиром да апликација има .env фајл, који није доступан на GitHub-у. Можемо сматрати да је овај фактор испуњен.

 **Помоћни сервиси (Backing services)**:

* **Испуњено? ✅**
* Објашњење: Спољни сервиси као што су базе података (MongoDB), SMTP сервери (Nodemailer) и API-ји (Twilio) третирају се као прикључни ресурси који се могу лако заменити или скалирати. Можемо сматрати да је овај фактор испуњен.

 **Изградња, издавање, покретање (Build, release, run)**:

* **Испуњено?** ❓
* Објашњење: Процеси изградње, објављивања и покретања апликације морају бити раздвојени, што би омогућило јасно дефинисање корака за deployment и смањење могућности грешака. Апликација има одвојене процесе за изградњу и покретање, али се не објављује нигде, обзиром да се не налази ни на каквом серверу. Овде постоји додатан простор за напредак, обезбеђивањем контејнера за апликацију.

 **Процеси (Processes)**:

* **Испуњено? ✅**
* Објашњење: Апликација је дизајнирана као скуп stateless процеса који могу лако да се скалирају хоризонтално и замене у случају неуспеха, што значи да сваки процес може бити прекинут и поново покренут без губитка података (подаци се чувају у базама). Можемо сматрати да је овај фактор испуњен.

 **Повезивање порта (Port binding)**:

* **Испуњено? ✅**
* Објашњење: Како Express.js експонира своје апликације путем повезивања портова, сервиси унутар апликације су тако експонирани, нпр. HTTP сервер слуша на одређеном порту који се може конфигурисати, па тако можемо сматрати да је овај фактор испуњен.

 **Конкурентност (Concurrency)**:

* **Испуњено?** ❓
* Објашњење: Node.js процеси могу бити скалирани хоризонтално помоћу кластера или контејнера. Апликација може бити дизајнирана за скалирање путем конкурентних процеса, чиме се оптимизују ресурси и перформансе, тако да овај фактор није испуњен, али је јасно како можемо доћи до његовог испуњења.

 **Једноставна замена (Disposability)**:

* **Испуњено? ✅**
* Објашњење: Node.js апликације обично брзо стартују. Према томе, наша апликација брзо стартује и исто тако брзо се гаси, што побољшава робусност апликације и омогућава лакше управљање у случају непланираних прекида. Можемо сматрати да је овај фактор испуњен.

 **Паритет развоја и продукције (Dev/prod parity)**:

* **Испуњено?** ❓
* Објашњење: Користећи Docker, може се обезбедити да развојна и продукциона окружења буду слична. Окружења за развој, тестирање и продукцију морају бити што сличнија, што смањује проблеме који произилазе из разлика у конфигурацији и окружењу. Овај фактор апликације није испуњен.

 **Логови (Logs)**:

* **Испуњено?** ❓
* Објашњење: Логовање треба да буде централизовано и третирано као ток догађаја који се могу анализирати. Овај фактор није испуњен, али се може испунити коришћењем лог сервиса попут Loggly, ELK Stack итд.

 **Административни процеси (Admin processes)**:

* **Испуњено? ✅**
* Објашњење: Административни задаци, као што су управљање базом података или извршавање миграција, изоловани су и извршавају се као једнократни процеси. У нашој апликацији не постоје имплементиране миграције, али управљање базом података је изолован процес, па можемо сматрати да је овај фактор испуњен.

Апликација за управљање библиотеком је веома добро дизајнирана и испуњава многе од захтева 12-факторске методологије, али постоји неколико области у којима се могу направити побољшања. Испуњено је 7 од 12 жељених фактора. Веб апликација има јединствени код, али мобилна апликација има свој засебан код. За побољшање, треба размотрити могућности за дељење кода или коришћење микро-сервисне архитектуре. Процеси изградње и покретања су одвојени, али апликација није објављена на серверу. Имплементација CI/CD и контејнеризација би олакшале издавање и покретање апликације и покриле неколико неиспуњених фактора. Иако постоји потенцијал за хоризонтално скалирање, он није у потпуности искоришћен. Стога, могу се искористити кластери или контејнери за боље управљање ресурсима и побољшање перформанси. Окружења за развој и продукцију нису довољно усклађена, па би коришћење Docker-а обезбедило сличност између њих. Успостављање ових побољшања ће додатно унапредити робусност, скалабилност и одрживост апликације, приближавајући је идеалима 12-факторске методологије.