

# Флак

## Web Девелормент

---

РАЗВОЈ ВЕБ АПЛИКАЦИЈА СА ПИТХОН-ОМ

Мигуел Грин

ерг

# Flask Web Development

Take full creative control of your web applications with Flask, the Python-based microframework. With the second edition of this hands-on book, you'll learn Flask from the ground up by developing a complete, real-world application created by author Miguel Grinberg. This refreshed edition accounts for important technology changes that have occurred in the past three years.

Explore the framework's core functionality, and learn how to extend applications with advanced web techniques such as database migrations and an application programming interface. The first part of each chapter provides you with reference and background for the topic in question, while the second part guides you through a hands-on implementation.

If you have Python experience, you're ready to take advantage of the creative freedom Flask provides. Three sections include:

- **A thorough introduction to Flask:** explore web application development basics with Flask and an application structure appropriate for medium and large applications
- **Building Flasky:** learn how to build an open source blogging application step-by-step by reusing templates, paginating item lists, and working with rich text
- **Going the last mile:** dive into unit testing strategies, performance analysis techniques, and deployment options for your Flask application

“The second edition upholds the strong tradition of Miguel's blog posts and the book's first edition that, together, fueled my learnings about Flask, including database interactions and deployments.”

—Jason Myers

author, *Essential SQLAlchemy*,  
2nd Edition (O'Reilly)

---

**Miguel Grinberg** has over 25 years of experience as a software engineer. He blogs at <https://blog.miguelgrinberg.com> about a variety of topics including web development, Python, robotics, photography, and the occasional movie review.

US \$44.99

CAN \$59.99

ISBN: 978-1-491-99173-2



5 4 4 9 9  
9 781491 991732



Twitter: @oreillymedia  
[facebook.com/oreilly](http://facebook.com/oreilly)

ДРУГО ИЗДАЊЕ

---

# Фласк ве развој

## Развој ве апликација са Питхон-ом

Мигел Грин ерг

Пекин•Бостон•Фарнхам•Севастопол•Токио

O'REILLY®

Флакс Ве Девелопмент

Мигуел Грин ерг

Ауторско право © 2018 Мигуел Грин ерг. Сва права задржана.

Штампано у Сједињеним Америчким Државама.

О јавио О'Reilly Медиа, Инц., 1005 Гравенстеин Хигхвей Нортх, Сејстопол, ЦА 95472.

О'Reilly књиге се могу купити за о разовну, пословну или промотивну употребу. Онлайн издања су такође доступна за вебину наслова (<http://oreilly.com/safari>). За више информација, контактирајте наше корпоративно/институционално одељење продаје: 800-998-9938 или [corporate@oreilly.com](mailto:corporate@oreilly.com).

Уредник: Алисон Мекдоналд

Индекс: Еллен Троутман

Уредник продукције: Цоллеен Цоле

Дизајнер ентеријера: Давид Футато

Уредник копије: Двайт Ремзи

Дизајнер омота: Ранди Џомер

Лектор: Рацхел Хеад

Илустратор: Ребека Демарест

март 2018: Друго издање

Историја ревизија за друго издање

2018-03-02: Прво издање

Погледајте <http://oreilly.com/catalog/errorata.cgi?isbn=9781491991732> за детаље о издању.

О'Reilly лого је регистровани заштитни знак компаније O'Reilly Медиа, Инц. Флакс Ве Девелопмент, насловна слика и повезана трговачка одећа су заштитни знаци O'Reilly Медиа, Инц.

Иако су издавач и аутор уложили напоре у доје да ове еде да су информације и упутства садржана у овом делу тачна, издавач и аутор се одричу сваке одговорности за грешке или пропусте, укључујући даје ограничења одговорност за штету која настане услед коришћења или ослањање на ово дело. Коришћење информација и упутстава садржаних у овом раду је на сопствени ризик. Ако је илјади пример кода или друга технологија коју ово дело садржи или описује подлеже лиценцама отвореног кода или правима интелектуалне својине других, ваша је одговорност да ове еде да је ваша употреба у складу са таквим лиценцама и/или правима.

978-1-491-99173-2

[ЛСИ]

---

# Преглед садржаја

Предговор.....	КИ
----------------	----

---

## Део И. Увод у Фласк

1. Инсталација.....	1
Креирање виртуелних окружења директоријума апликација	2
Креирање виртуелног окружења са Питхон 3	3
Креирање виртуелног окружења са Питхон 2	3
Рад са виртуелним окружењем	4
Инсталирање Питхон пакета са пип-ом	5
2. Основна структура апликације.....	7
Иницијализација	7
Руте и функције прегледа	8
Комплетна апликација	9
Ве сервер за развој	10
Динамиц Роутес	12
Режим за отклањање грешака	13
Опције командне линије	15
Циклус захтев-одговор	17
Контексти апликације и захтева	17
Рекуест Диспатџинг	18
Тхе Рекуест О јеџт	19
Рекуест Хоокс	20
Одговори	21
Фласк Ектенсионс	23

3. Ша лони.....	25
Тхе Јиња2 Темплате Енгине	26
Рендеринг Темплатес	26
Променљиве	27
Контролне структуре	28
Боотстрап интеграција са Флакс-Боотстррап-ом	30
Странице прилагођених грешака	33
Линкови	36
Статиц Филес	37
Локализација датума и времена са флакс-моментом	38
4. Ве о расци.....	43
Конфигурација	44
Форм Џласses	44
ХТМЛ приказивање о разаца	47
Руковање о расцима у функцијама приказа	48
Преусмеравања и корисничке сесије	51
Порука трепери	53
5. Базе података.....	57
СКЛ азе података	57
НоСКЛ азе података	58
СКЛ или НоСКЛ?	59
Питхон Дата асе Фрамеворкс	59
Управљање азом података са Флакс-СКЛАлцхеми Модел	61
Дефиниција Релације Операције азе података Креирање	62
та ела Уметање редова Измена редова Брисање редова	64
Постављање упита редова	66
	66
	68
	68
	68
Коришћење азе података у функцијама приказа	71
Интеграција са Питхон Схелл-ом	72
Миграције азе података са Флакс-Миграте	73
Креирање миграционог спремишта	73
Креирање скрипте за миграцију	74
Надоградња азе података	75
Додавање више миграција	76

6. Емайл. . . . .	79
Подршка путем е-поште са Флакс-Майл	79
Слање е-поште из Питхон Схелл-а	81
Интеграција е-поште са апликацијом	81
Слање асинхроне е-поште	83
7. Велика структура апликације. . . . .	85
Опције конфигурације	85
структуре пројекта Пакет	86
апликације Коришћење	88
фа рике апликација	88
Имплементација функционалности апликације у нацрту	90
Аплицијацион Скрипт	93
Рекуриментс Филе	93
Јединични тестови	94
Подешавање азе података	96
Покретање апликације	97
<hr/>	
ИИ део. Пример: Апликација за друштвено логовање	
8. Аутентификација корисника. . . . .	101
Проширење за провјеру аутентичности за	101
Флакс Сигурност лозинке Хеширање лозинки	102
са Веркеуг-ом Креирање нацрта за	102
аутентификацију Аутентификација корисника	105
помоћу Флакс-Логин-а Припрема корисничког	107
модела за пријаве Заштита ruta Додавање	107
о расца за пријаву Потписивање корисника	108
Пријављивање корисника Разумијевање како	109
Флакс-Логин функционира Тестирање пријава	111
Додавање регистрације нових корисника	112
О разац за регистрацију корисника за	113
регистрацију нових корисника	114
	115
	115
	117
Потврда налога	118
Генерирање токена за потврду са својим опасним	118
Слање е-порука за потврду	120
Управљање налогом	125

9. Улоге корисника.....	127
Репрезентација улога у ази података	127
Верификација улога	131
	132
10. Усер Про лес.....	137
Информације о профилу	137
Страница корисничког профила	138
Уређивач профила	141
Уређивач профила на нивоу корисника	141
Уређивач профила на нивоу администратора	143
Усер Аватарс	146
11. Блог Постови.....	151
Слање и приказ о јаве на логу	151
Блог постови на страницама профила	154
Пагинирање других листа лог постова	155
Креирање лажних података о о јавама	155
на логу Рендеровање на страницама	157
Добавање виџета за пагинацију	158
Постови о огађеног текста са Маркдован и Флакс-Пагедован	161
Коришћењем Флакс-Пагедован	162
Руковање о огађеним текстом на серверу	164
Сталне везе до постова на логу	165
Уредник логова	167
12. След еници.....	171
Односи азе података Ревидирани	171
односи „много-према-више“	172
Референтни односи само-референције	174
Напредни односи „много-према-више“ Пратиоци	174
на страници профила Постављање упита праћених	178
постова помоћу придруживања ази података Приказ	181
праћених постова на почетној страници	183
13. Коментари корисника.....	189
Представљање коментара у ази података	189
Подношење коментара и приказ модерирања	191
коментара	193
14. Интерфејси за програмирање апликација.....	199
Увод у РЕСТ	199

Ресурси су све	200
Рекуест Методс	201
Органи за тражење и одговор	201
Версионинг	202
РЕСТфул ве услуге са фласком	203
Креирање АПИ плана	203
Грешка руковање	204
Аутентификација корисника помоћу Флакс-ХТТПАутх	206
Аутентификација заснована на токенима	208
Серијализација ресурса у и из JSON-а	210
Имплементација крајњих тачака ресурса	213
Пагинација великих колекција ресурса	216
Тестирање ве услуга са ХТТПие-ом	217
<hr/>	
<b>ИИИ део. Последња миља</b>	
<b>15. Тестирање.....</b>	<b>221</b>
До ијање извештаја о покривености	221
кода Флак тест Клијентско тестирање	224
ве апликација Тестирање ве	225
услуга Свео ухватно тестирање са	228
селеном Да ли је вредно тога?	230
	234
<b>16. Перформансе.....</b>	<b>237</b>
Евидентирање спорих перформанси азе података	237
Профилисање извornог кода	239
<b>17. Распоређивање.....</b>	<b>241</b>
Ток рада имплементације	241
Евидентирање грешака током	242
имплементације производног о лака	243
Хероку платформа	244
Припрема апликације	244
Тестирање са Хероку Лоцал-ом	253
Примена помоћу гит пусх-а	254
Примена надоградње	255
Доцкер контејнери	256
Инсталирање Доцкер-а	256
Прављење слике контејнера	257
Покретање контејнера	261

Провера радног контејнера	262
Гурање слике вашег контејнера у спољни регистар	263
Коришћење екстернег азе података	264
Оркестрација контејнера са Доцкер Цомпосе	265
Чишћење старих контејнера и слика	269
Коришћење Доцкер-а у производњи	270
Традиционалне примене	270
Подешавање сервера	271
Увоз варија или окружења	271
Подешавање евидентирања	272
<b>18. Додатни ресурси . . . . .</b>	<b>275</b>
Коришћење интегрисаног развојног окружења (ИДЕ)	275
Приказивање флаја екстензија	276
До ијање помоћи	276
Укључивање у Флај	277
Индекс . . . . .	279

---

## Предговор

Флакс се издава од других оквира јер омогуѓава програмерима да заузму возачко место и имају пуну креативну контролу над својим апликацијама. Можда сте раније чули фразу „ор а против оквира“. Ово се дешава са већином оквира када одлучите да решите про лем помоћу решења које није званично. Може ити да желите да користите другачији механизам азе података или можда другачији метод аутентификације корисника. Одступање од путање које су поставили програмери оквира задаће вам много главо оља.

Флакс није такав. Да ли волите релационе азе података? Велики. Флакс их све подржава. Можда више волите НоСКЛ азу података? Нема про лема. Флакс такође ради са њима. Желите да користите сопствени домаћи механизам за азе података? Уопште вам није потреба на аза података? Још увек до ро. Са Флакс-ом можете ода рати компоненте своје апликације или чак написати сопствену ако је то оно што желите. Без питања!

Кључ ове слојде је у томе што је Флакс од самог почетка дизајниран да уде проширен. Долази са ројусним језгром које укључује основну функционалност која је потребна свим већим апликацијама и очекује да остатак о језиди нека од многих екstenзија трећих страна у екосистему – и, наравно, ви.

У овој књизи представљам свој ток рада за развој веће апликација са Флаксом. Не тврдим да је ово једини прави начин за прављење апликација са овим оквиром. Трејало и да схватите моје изоре као препоруке, а не као јеванђеље.

Већина књига о развоју софтвера пружа мале и фокусиране примере кода који демонстрирају различите карактеристике циљне технологије у изолацији, остављајући „лепљиви“ код који је неопходан да се ове различите карактеристике трансформишу у потпуно функционалну апликацију коју читалац може да схвата. Ја имам потпуно другачији приступ. Сви примери које представљам део су једне апликације која почиње веома једноставно и проширује се у сваком следећем поглављу. Ова апликација почиње живот са само неколико линија кода и завршава се као лепо представљена апликација за логовање и друштвене мреже. ција.

## За кога је ова књига

Тре ало и да имате одређени ниво искуства у Питхон кодирању да исте максимално искористили ову књигу. Иако књига претпоставља да нема претходног Фласк знања, претпоставља се да су концепти Пајтона као што су пакети, модули, функције, декоратори и о јектно оријентисано програмирање до ро схваћени. Познавање изузетака и дијагностиковања про лема из праћења стека иће веома корисно.

Док радите кроз примере у овој књизи, провешћете много времена у командној линији. Тре ало и да се осећате пријатно користећи командну линију свог оперативног система.

Модерне ве апликације не могу да из егну употребе у ХТМЛ-а, ЦСС-а и ЈаваСкрипт-а. Пример апликације који је развијен у целиот књизи очигледно користи ово, али сама књига не улази у много детаља у вези са овим технологијама и начином на који се оне користе. Одређени степен познавања ових језика се препоручује ако намеравате да развијете комплетне апликације вез помоћи програмера који је упућен у технике на страни клијента.

О јавио сам пратећу апликацију за ову књигу као отворени код на ГитХу -у. Иако ГитХу омогућава преузимање апликација као очичних ЗИП или ТАР датотека, топло препоручујем да инсталirate Гит клијент и да се упознате са контролом верзија извornog кода (дрем са основним командама за клонирање и проверу различитих верзија апликације директно из складишта). Кратка листа команда које ће вам треати приказана је у [одељку „Како радити са примером кода“ на страни кии.](#) Пожелете да користите контролу верзија и за своје пројекте, па користите ову књигу као изговор да научите Гит!

Конечно, ова књига није потпуна и исцрпна референца на Фласк оквир. Већина функција је покривена, али ову књигу и тре ало да допуните [званичном документацијом за Фласк.](#)

## Како је ова књига организована

Ова књига је подељена на три дела.

[Први део, Увод у Фласк,](#) истражује основе развоја ве апликација са Фласк оквиром и неким његовим проширењима:

- [Поглавље 1](#) описује инсталацију и подешавање Фласк оквира.
- [Поглавље 2](#) улази директно у Фласк са основном апликацијом.
- [Поглавље 3](#) уводи употребе у ша лона у Фласк апликацијама.
- [Поглавље 4](#) представља ве форме.
- [Поглавље 5](#) уводи азе података.

- Поглавље 6 уводи подршку путем е-поште. •

Поглавље 7 представља структуру апликације која је прикладна за средње и велике апликације.

Део ИИ, Пример: Апликација за друштвено логовање, гради Фласки, апликацију за логовање и друштвене мреже отвореног кода коју сам развио за ову књигу:

- Поглавље 8 имплементира систем аутентификације корисника.
- Поглавље 9 имплементира корисничке улоге и дозволе. •

Поглавље 10 имплементира странице корисничког профилла. •

Поглавље 11 креира интерфејс за логовање. • Поглавље 12

имплементира след енике. • Поглавље 13 имплементира

коментаре корисника за постове на логу. • Поглавље 14

имплементира интерфејс за програмирање апликације (АПИ).

Део ИИИ, Последња миља, описује неке важне задатке који нису директно повезани са кодирањем апликације које треба размотрити пре о јављивања апликације:

- Поглавље 15 детаљно описује различите стратегије јединичног тестирања.
- Поглавље 16 даје преглед техника анализе учинка. • Поглавље 17 описује опције примене за Флакс апликације, укључујући традиционалне ционална решења заснована на о лаку и контејнере.

• Поглавље 18 наводи додатне ресурсе.

## Како радити са примером кода

Примери кода представљени у овој књизи доступни су за преузимање на [хттпс://гитху.цом/мигуелгин/ерг/флакси](https://github.com/miguelgrin/erg/flask).

Историја урезивања у овом спремишту је пажљиво креирана да и одговарала редоследу којим су концепти представљени у књизи. Препоручени начин рада са кодом је да проверите урезивање почевши од најстаријег, а затим идете напред кроз листу урезивања док напредујете са књигом. Као алтернатива, Гит-Ху ће вам такође омогућити да преузмете свако урезивање као ЗИП или ТАР датотеку.

Ако одлучите да користите Гит за рад са изврним кодом, онда морате да инсталirate Гит клијент, који можете преузети са [хттп://гит-см.цом](https://git-cm.com). Следећа команда преузима пример кода користећи Гит:

```
$ гит клон хттпс://гитху.цом/мигуелгин/ерг/флакси.гит
```

Команда гит цлоне инсталира изворни код са ГитХу -а у фасцикул фласки2 која је креирана у тренутном директоријуму. Ова фасцикула не садржи само изворни код; копија Гит спремишта са целокупном историјом промена направљених у апликацији је такође укључена.

У првом поглављу од вас ће ити затражено да погледате почетно издање апликације, а затим, на одговарајућим местима, ићете упућени да идете напред у историји.

Гит команда која вам омогућава да се крећете кроз историју промена је гит цхецкоут.

Ево примера:

```
$ git цхецкоут 1a
```

1а наведен у команди је ознака : именована тачка у историји урезивања пројекта. Ово спремиште је означену у складу са поглављима књиге, тако да ознака 1а која се користи у примеру поставља датотеке апликације на почетну верзију која се користи у **поглављу 1**. Већина поглавља има више од једне ознаке повезане са њима, тако да нпр., ознаке 5а, 5 и тако даље су инкременталне верзије представљене у **поглављу 5**.

Када покренете гит цхецкоут команду као што је управо приказано, Гит ће приказати поруку упозорења која вас оавештава да сте у стању „одвојеног ГЛАВА“. То значи да нисте ни у једној специфичној грани кода која може да прихвати нова урезивања, већ уместо тога гледате одређено урезивање у средини историје промена пројекта.

Нема разлога да удете узнемирени због ове поруке, али трејте да имате на уму да ако извршите измене или које датотеке док су у овом стању, издавање још једног гит цхецкоут -а неће успети, јер Гит неће знати шта да ради са променама ти си направио.

Дакле, да исте могли да наставите да радите са пројектом, мораћете да вратите датотеке које сте променили у прво итно стање. Најлакши начин да то урадите је помоћу гит ресет команде:

```
$ git ресет --хард
```

Ова команда ће уништити све локалне промене које сте направили, тако да и трејте ало да сачувате све што не желите да изгубите пре него што употребите ову команду.

Поред провере изворних датотека за верзију апликације, у одређеним тренуцима ћете морати да извршите додатне задатке подешавања. На пример, у неким случајевима мораћете да инсталirate нове Питхон пакете или примените ажурирања на азу података. Биће вам речено када су то неопходне.

С времена на време, можда ћете желети да освежите своје локално спремиште са оног на ГитХу -у, где су можда примењене исправке грешака и пољашања. Команде које то постижу су:

```
$ git дохвати --све  
$ git дохвати --ознаке  
$ git ресет --хард оригинал/мастер
```

Команде гит фетцх се користе за ажурирање историје урезивања и ознака у вашем локалном спремишту са удаљеног на ГитХу -у, али ништа од тога не утиче на стварне изворне датотеке, које се ажурирају гит ресет командом која следи.

Још једном, имајте на уму да ћете сваки пут када се користи гит ресет изгу ити све локалне промене које сте направили.

Још једна корисна операција је да видите све разлике између две верзије апликације. Ово може ити веома корисно за детаљно разумевање промене. Из командне линије, команда гит дифф то може да уради. На пример, да видите разлику између ревизија 2а и 2 , користите:

```
$ git diff 2a 2
```

Разлике су приказане као закрпа, што није аш интуитиван формат за преглед промена ако нисте навикили да радите са датотекама закрпа. Можда ћете открити да су графичка поређења која показује ГитХу много лакша за читање. На пример, разлике између ревизија 2а и 2 могу се видети на [ГитХу -у на `https://gitHub.com/miguelgrin/erg/flask/compare/2a...2`](https://gitHub.com/miguelgrin/erg/flask/compare/2a...2).

## Коришћење примера кода

Ова књига је овде да вам помогне да завршите свој посао. Генерално, ако се уз ову књигу нуди пример кода, можете га користити у својим програмима и документацији. Не морате да нас контактирате за дозволу осим ако не репродукујете значајан део кода. На пример, за писање програма који користи неколико делова кода из ове књиге није потре на дозвола. Продаја или дистриуција ЦД-РОМ-а са примерима из О'Реилли књига захтева дозволу. Да исте одговорили на питање цитирањем ове књиге и цитирањем примера кода, није потре на дозвола. Уграђивање значајне количине примера кода из ове књиге у документацију вашег производа захтева дозволу.

Ми ценимо, али не захтевамо, приписивање. Атриуција о ично укључује наслов, аутора, издавача и ИСБН. На пример: „Флакс Ве Девелопмент, 2нд Едитион, Мигуел Грин ерг (О'Реилли). Ауторска права 2018 Мигуел Грин ерг, 978-1-491-99173-2.”

Ако сматрате да је ваша употреба примера кода ван поштене употребе или горе наведене дозволе, сло одно нас контактирајте на [permissions@oreilly.com](mailto:permissions@oreilly.com).

## Конвенције које се користе у овој књизи

У овој књизи се користе следеће типографске конвенције:

Курзив

Означава нове термине, УРЛ адресе, адресе е-поште, називе датотека и екstenзије датотека.

#### Константна ширина

Користи се за излаз командне линије и листе програма, као и унутар параграфа за упућивање на команде и програмске елементе као што су имена променљивих или функција, азле података, типови података, променљиве окружења, изјаве и кључне речи.

#### Константна ширина

поде љано Приказује команде или други текст који корисник тре да укуца дословно.

#### Курзивне или угаоне заграде константне ширине (<>)

Означава текст који тре а заменити вредностима које је унео корисник или вредностима одређеним контекстом.



Овај елемент означава савет или предлог.



Овај елемент означава општу напомену.



Овај елемент указује на упозорење или опрез.

#### О'Реилли Сафари



Сафари (раније Сафари Bookс Онлайн) је о ука заснована на чланству и референтна платформа за предузећа, владу, едукаторе и појединце.

Чланови имају приступ хиљадама књига, видео снимака за о уку, путева учења, интерактивних туторијала и курираних плејлиста од преко 250 издавача, укључујући О'Реилли Медиа, Харвард Бусинес Ревиев, Прентице Халл Професионал, Адисон-Весли Професионал, Мицрософт Прес , Самс, Кје, Пеацхпит Прес, Адо е, Фоџал Прес, Цисцо Прес, Јохн Вилеи & Сонс, Сингресс, Морган Кауфманн, ИБМ Ред оокс, Пацкт, Адо е Прес, ФТ Прес, Апресс, Маннинг, Нев Ридерс, МцГрав-Хилл, Јонес & Бартлетт и Цоурсе Технологи, између осталих.

За више информација посетите [хттп://ореилли.ком/сафари](http://oreillli.com/сафари).

## Како да нас контактирате

Молимо да коментаре и питања у вези са овом књигом упутите издавачу:

О'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Santa Rosa, CA 95472 800-998-9938 (у  
Сједињеним Државама или Канади) 707-829-0515  
( међународни или локални) 707-829-0104 (факс)

Имамо ве страницу за ову књигу, где наводимо грешке, примере и све додатне информације. Овој страници можете приступити на [хттп://ит.ли/флакс-ве-дев2](http://ит.ли/флакс-ве-дев2).

Да исте коментарисали или поставили техничка питања о овој књизи, пошаљите е-пошту на [оаккјес-тионс@ореилли.ком](mailto:oakkyes-tionc@oreillli.com).

За више информација о нашим књигама, курсевима, конференцијама и новостима, погледајте нашу ве страницу на [хттп://ввв.ореилли.ком](http://www.oreillli.com).

Пронађите нас на Фејс уку: [хттп://фаце.оок.ком/ореилли](http://фаце.оок.ком/ореилли)

Пратите нас на Твиттеру: [хттп://твиттер.ком/ореиллимедиа](http://твиттер.ком/ореиллимедиа)

Гледајте нас на Јоутубу е-у: [хттп://ввв.иоутуб.е.ком/ореиллимедиа](http://ввв.иоутуб.е.ком/ореиллимедиа)

## Признања

Не их могао сам написати ову књигу. До ио сам велику помоћ од породице, сарадника, старих пријатеља и нових пријатеља које сам стекао успут.

Желео их да се захвалим Брендану Кохлеру на његовом детаљном техничком прегледу и на његовој помоћи у олковању поглавља о интерфејсима за програмирање апликација. Такође дугујем Дејвиду Баумголду, Тоду Брунхофу, Сесилу Року и Метју Хјугу, који су прегледали рукопис у различитим фазама доворшавања и дали ми веома корисне савете у вези са тим шта да покријем и како да организујем материјал.

Писање примера кода за ову књигу представљало је значајан напор. Ценим помоћ Данијела Хофмана, који је урадио детаљан преглед кода апликације и указао на неколико по олшања. Такође сам захвалан свом сину тинејџеру, Дилану Гринергу, који је суспендовао своју зависност од Минецрафт-а на неколико викенда и помогао ми да тестирам код на неколико платформи.

О'Reilly има диван програм под називом Еарли Релеасе који омогућава нестрпљивим читаоцима да имају приступ књигама док се пишу. Нека од мојих читања о раном пуштању-

Они су отишли даље и упустили се у корисне разговоре о свом искуству радећи кроз књигу, што је довело до значајних по ольшања. Желео их да се посе но захвалим Сундипу Гупти, Дену Керону, Брајану Вистију и Кодију Скоту за доприносе који су дали овој књизи.

Осо ље О'Reилли Медиа је увек ило ту за мене. Изнад свега, желео их да одам признање мојој дивној уредници, Меган Бланшет, за њену подршку, савете и помоћ од првог дана нашег сусрета. Мег је искуство писања моје прве књиге учинила неза оправним.

Да закључим, желео их да се захвалим одличној Фласк заједници.

## Додатне хвале за друго издање

Желео их да се захвалим Али Мекдоналд, мојој уредници за друго издање ове књиге, као и Сузан Конант, Рејчел Румелиотис и целом тиму у O'Reilly Media на њиховој континуираној подршци.

Технички рецензенти за ово издање су урадили диван посао указујући на оласти које тре а по ольшати и пружајући ми нове перспективе. Желео их да одам признање Лорени Меси, Дајан Чен и Џеси Смит за њихов велики допринос кроз њихове повратне информације и сугестије. Такође веома ценим помоћ магистра сина, Дилана Грин ерга, који је мукотрпно тестирао све примере кода.

ДЕО И

---

## Увод у Фласк

## ПОГЛАВЉЕ 1

# Инсталација

Фласк је мали оквир према већини стандарда – довољно мали да се назове „микро оквир“, и довољно мали да ћете, када се упознавате са њим, вероватно моћи да прочитате и разумете сав његов изворни код.

Али то што је мали не значи да ради мање од других оквира. Фласк је дизајниран као прошириви оквир од темеља; пружа солидно језгро са основним услугама, док проширења пружају остало. Пошто можете да ирате и ирате пакете проширења које желите, на крају ћете до ити леан стацк који нема надувавање и ради управо оно што вам је потре но.

Фласк има три главне зависности. Подсистеми за рутирање, отклањање грешака и интерфејс мрежног пролаза сервера (ВСГИ) долазе од Веркзеуга ; подршку за ша лоне о ез еђује Јиња<sup>2</sup>; а интеграција командне линије долази од Цлицк. Све ове зависности је аутор Армин Ронацхер, аутор Фласк-а.

Фласк нема изворну подршку за приступ азама података, валидацију ве о разаца, аутентификацију корисника или друге задатке високог нивоа. Ове и многе друге кључне услуге које су већини ве апликација потре не доступне су преко екстензија које се интегришу са основним пакетима. Као програмер, имате моћ да иза ерете екстензије које нај оље функционишу за ваш пројекат, или чак да напишете сопствена ако осећате да сте склони. Ово је у супротности са већим оквиром, где је већина из ора направљена за вас и које је тешко или понекад немогуће променити.

У овом поглављу ћете научити како да инсталirate Фласк. Једини услов је рачунар са инсталirаним Питхон-ом.



Примери кода у овој књизи су верификовани да раде са Питхоном 3.5 и 3.6. По жељи се може користити и Питхон 2.7, али с о зиром на то да ова верзија Питхона неће бити одржавана након 2020. године, топло се препоручује да користите верзије 3.к.



Ако планирате да користите Мицрософт Виндовс рачунар за рад са примером кода, морате да одлучите да ли желите да користите „матични“ приступ заснован на Виндовс алатима или да подесите рачунар на начин који вам омогућава да усвојите више главни скуп алата заснован на Уник. Код представљен у овој књизи је у великој мери компатибилан са овим приступом. У неколико случајева у којима се приступи разликују, следи се Уник решење и наводе се алтернативе за Виндовс.

Ако одлучите да пратите Уник ток посла, имате неколико опција. Ако користите Виндовс 10, можете да омогућите Виндовс подсистем за Линук (ВСЛ), што је званично подржана функција која креира Униту Линук инсталацију која ради уз изворни Виндовс интерфејс, дајући вам приступ асах љусци и комплетан сет алата заснованих на Унику. Ако ВСЛ није доступан на вашем систему, друга доја опција је [Цигвин](#), пројекат отвореног кода који емулира ПОСИКС подсистем који користи Уник и обезбеђује портове великог броја Уник алати.

## Креирање директоријума апликација

За почетак, потребно је да креирате директоријум који ће угостити пример кода, који је доступан у ГитХу спремишту. Као што је описано у [одељку „Како радити са примером кода“ на страници Унике](#), најпогоднији начин да то урадите је да проверите код директно са ГитХу -а користећи Гит клијент. Следеће команде преузимају пример кода са ГитХу -а и иницијализују апликацију на верзију 1а, што је почетна верзија са којом ћете радити:

```
$ git клон хттп://гитху.цом/мигуелгрин/ерг/фласки.гит $ цд  
фласки $ git цхецкоут 1а
```

Ако не желите да користите Гит и уместо тога ручно откуцате или копирате код, можете једноставно да креирате празан директоријум апликације на следећи начин:

```
$ мкдир фласки  
$ цд фласки
```

## Виртуелна окружења

Сада када сте креирали директоријум апликације, време је да инсталirate Флакс. Најпогоднији начин да то урадите је коришћење виртуелног окружења. Виртуелно окружење

је копија Питхон интерпретера у који можете приватно инсталаријати пакете, аз утицаја на гло ални Питхон интерпретер инсталариран у вашем систему.

Виртуелна окружења су веома корисна јер спречавају неред у пакетима и сукоб између верзија у системском Питхон интерпретатору. Креирање виртуелног окружења за сваки пројекат осигуруја да апликације имају приступ само пакетима које користе, док гло ални тумач остаје уредан и чист и служи само као извор из којег се може креирати више виртуелних окружења. Као додатна предност, за разлику од Питхон тумача за читав систем, виртуелна окружења се могу креирати и њима управљати аз администраторских права.

## Креирање виртуелног окружења са Питхон 3

Стварање виртуелних окружења је о ласт у којој се Питхон 3 и Питхон 2 тумачи разликују. Са Питхон-ом 3, виртуелна окружења су извршно подржана венв пакетом који је део стандардне и лиотеке Питхон-а.



Ако користите стандардни Питхон 3 тумач на Униту Линук систему, стандардни венв пакет није подразумевано инсталариран. Да исте га додали у свој систем, инсталирајте питхон3-венв пакет на следећи начин:

```
$ sudo apt-get install python3-virtualenv
```

Команда која креира виртуелно окружење има следећу структуру:

```
$ python3 -m venv имя виртуелног окружења
```

Опција -м венв покреће венв пакет из стандардне и лиотеке као самосталну скрипту, прослеђујући жељено име као аргумент.

Сада ћете креирати виртуелно окружење унутар фласки директоријума. Уо ичажена конвенција за виртуелна окружења је да их назовете венв, али можете користити и друго име ако желите. Уверите се да је ваш тренутни директоријум подешен на фласки, а затим покрените ову команду:

```
$ python3 -m venv венв
```

Након што се команда заврши, имаћете поддиректоријум са именом венв унутар фласки, са потпуно новим виртуелним окружењем које садржи Питхон тумач за ексклузивну употребу у овом пројекту.

## Креирање виртуелног окружења са Питхон 2

Питхон 2 нема венв пакет. У овој верзији Питхон интерпретера, виртуелна окружења се креирају помоћу услужног програма виртуаленв треће стране.

Уверите се да је ваш тренутни директоријум подешен на фласки, а затим користите једну од следеће две команде, у зависности од вашег оперативног система. Ако користите Линук или мацОС, команда је:

```
$ судо пип инсталл виртуаленв
```

Ако користите Мицрософт Виндовс, уверите се да сте отворили прозор командне линије користећи опцију „Покрени као администратор”, а затим покрените ову команду:

```
$ пип инсталл виртуаленв
```

Команда виртуаленв узима име виртуелног окружења као свој аргумент. Уверите се да је ваш тренутни директоријум подешен на фласки, а затим покрените следећу команду да исте креирали виртуелно окружење под називом венв:

```
$ виртуаленв венв  
Нова извршна датотека за питхон у венв/ ин/  
питхон2.7 Такође креирање извршне датотеке у венв/  
ин/питхон Инсталација сетуптоолс, пип, вхеел...готово.
```

Поддиректоријум са именом венв иће креиран у тренутном директоријуму и све датотеке повезане са виртуелним окружењем иће унутар њега.

## Рад са виртуелним окружењем

Када желите да почнете да користите виртуелно окружење, морате га „активирати“. Ако користите Линук или мацОС рачунар, можете активирати виртуелно окружење помоћу ове команде:

```
$ соурце венв/ ин/актив
```

Ако користите Мицрософт Виндовс, команда за активацију је:

```
$ венв\Скриптс\активате
```

Када је виртуелно окружење активирано, локација његовог Питхон интерпретатора се додаје променљивој окружења PATH у вашој тренутној командној сесији, која одређује где да тражите извршне датотеке. Да вас подсети да сте активирали виртуелно окружење, команда за активацију мења вашу командну линију тако да садржи назив окружења:

```
(венв) $
```

Након што је виртуелно окружење активирано, куцањем питхон у командној линији ће се позвати тумач из виртуелног окружења уместо тумача за читав систем. Ако користите више од једног прозора командне линије, морате да активирате виртуелно окружење у сваком од њих.



Док је активирање виртуелног окружења о ично најповољнија опција, виртуелно окружење можете користити и ез активирања. На пример, можете да покренете Питхон конзолу за венв виртуелно окружење тако што ћете покренути венв/ ин/питхон на Линук-у или мацОС-у или венв\Скриптс\питхон на Мицрософт Виндовс-у.

Када завршите са радом са виртуелним окружењем, откуцајте деацтивате на командној линији да исте вратили варија лу окружења ПАТХ за вашу терминалску сесију и командни редак у њихова оригинална стања.

## Инсталирање Питхон пакета са пип-ом

Питхон пакети се инсталирају са пип менаџером пакета, који је укључен у сва виртуелна окружења. Попут команде питхон , куцање пип у сесији командне линије ће позвати верзију овог алата која припада активираном виртуелном окружењу. ронмент.

Да исте инсталерили Флакс у виртуелно окружење, уверите се да је венв виртуелно окружење активирано, а затим покрените следећу команду:

```
(венв) $ пип инсталл флакс
```

Када извршите ову команду, пип неће само инсталирати Флакс, већ и све његове зависности. У ило ком тренутку можете да проверите који су пакети инсталирани у виртуелном окружењу помоћу команде пип фреезе :

```
(венв) $ пип фреезе
клицк==6.7 Флакс==0.12.2
итсдангероус==0.24
Јиња2==2.9.6
МаркупСафе==1.0
Веркзеуг==0.12.2
```

Излаз замрзавања пип - а укључује детаљне ројеве верзија за сваки инсталирани пакет. Бројеви верзија које до ијете вероватно ће се разликовати од оних приказаних овде.

Такође можете да проверите да ли је Флакс исправно инсталiran тако што ћете покренути Питхон интерпретер и покушати да га увезете:

```
(венв) $ питхон
>>> импорт флакс
>>>
```

Ако се не појаве грешке, можете се и честитати: спремни сте за следеће поглавље, где ћете написати своју прву ве апликацију.

## ПОГЛАВЉЕ 2

## Основна структура апликације

У овом поглављу ћете научити о различитим деловима Флакс апликације. Такође ћете написати и покренути своју прву већу апликацију Флакс.

### Иницијализација

Све Флакс апликације морају креирати инстанцу апликације. Већ сервер прослеђује све захтеве које прими од клијената овом објекту за руковање, користећи протокол који се зове Веб Сервер Гатавац Интерфаце (ВСГИ, изговара се „виз-гхее“). Инстанца апликације је објекат класе Флакс, описано креирана на следећи начин:

```
из flask импорт Flask апп  
= Flask(__name__)
```

Једини потребни аргумент за конструктор класе Флакс је име главног модула или пакета апликације. За већину апликација, Питхон-ова променљива `__name__` је исправна вредност за овај аргумент.



Аргумент `__name__` који се прослеђује конструктору Флакс апликације је извор за унесење међу новим Флакс програмерима. Флакс користи овај аргумент да одреди локацију апликације, што му заузврат омогућава да лоцира друге датотеке које су део апликације, као што су слике и шаллони.

Касније ћете научити сложеније начине иницијализације апликације, али за једноставне апликације то је све што је потребно.

## Руте и функције прегледа

Клијенти као што су ве претраживачи шаљу захтеве ве серверу, који их заузврат шаље инстанци апликације Флакс. Инстанца апликације Флакс мора да зна који код тре а да покрене за сваки тражени УРЛ, тако да чува мапирање УРЛ-ова у Питхон функције. Повезаност између УРЛ-а и функције која њиме рукује назива се пута.

Најпогоднији начин да се дефинише пута у Флакс апликацији је преко ап.роуте декоратора који је изложен инстанци апликације. Следећи пример показује како се пута декларише коришћењем овог декоратора:

```
@апп.роуте('/')
дeф индeк():
    рeтурн '<x1>Здраво свет!</x1>'
```



Декоратори су стандардна карактеристика језика Питхон. Уо ичајена употреба декоратора је да региструју функције као функције руковаоца које се позивају када се додећи одређени догађаји.

Претходни пример региструје функцију индек() као руковаоца за основни УРЛ апликације. Док је ап.роуте декоратор префериран метод за регистровање функција прегледа, Флакс такође нуди традиционалнији начин за подешавање пута апликације помоћу методе апп.адд\_урл\_руле(), која у свом најосновнијем облику има три аргумента: УРЛ, назив крајње тачке и функција приказа. Следећи пример користи апп.адд\_урл\_руле() да региструје функцију индек() која је еквивалентна претходно приказаној:

```
дeф индeк():
    рeтурн '<x1>Здраво свет!</x1>'

апп.адд_урл_руле('/', 'индeк', индeк )
```

Функције као што је индек() које рукују УРЛ-овима апликација називају се функције приказа. Ако је апликација распоређена на серверу повезаном са именом домена ввв.екампле.цом, онда и навигација на [хттп://ввв.екампле.цом](http://ввв.екампле.цом) у вашем претраживачу покренула индек() да се покрене на серверу. Повратна вредност ове функције приказа је одговор који клијент доји. Ако је клијент ве претраживач, овај одговор је документ који се приказује кориснику у прозору претраживача. Одговор који враћа функција приказа може или једноставан стринг са ХТМЛ садржајем, али може имати и сложеније облике, као што ћете видети касније.



Уграђивање низова одговора са ХТМЛ кодом у Питхон изворне датотеке доводи до кода који је тешко одржавати. Примери у овом поглављу служе само за увођење концепта одговора. Научићете ољи начин за генерирање ХТМЛ одговора у 3. [поглављу](#).

Ако о ратите пажњу на то како се формирају неки УРЛ-ови за услуге које свакодневно користите, приметићете да многи имају променљиве секције. На пример, УРЛ за вашу Фаце оок страницу профилма има формат `http://www.фаце оок.цом/<ваше-име>`, који укључује ваше корисничко име, што га чини различитим за сваког корисника. Флак подржава ове типове УРЛ-ова користећи поседну синтаксу у декоратору `app.route`. Следећи пример дефинише руту која има динамичку компоненту:

```
@app.route('/user/<name>')
def user(name): return
'<x1>Здраво, {}!</x1>.format(name)
```

Део УРЛ-а руте у угаоним заградама је динамички део. Сви УРЛ-ови који одговарају статичким деловима иће мапирани на ову руту, а када се позове функција приказа, динамичка компонента ће ићи прослеђена као аргумент. У претходном примеру, аргумент `name` се користи за генерирање одговора који укључује персонализовани поздрав.

Динамичке компоненте у рутама су подразумевано низови, али такође могу ићи различитих типова. На пример, рута `/user/<инт:ид>` и одговарала само УРЛ адресама које имају цео број у динамичком сегменту ИД-а, као што је `/user/123`. Флак подржава типове стринг, инт, флоат и путању за руте. Тип путање је поседан тип стринга који може да садржи косе црте, за разлику од типа стринга.

## Комплетна апликација

У претходним одељцима научили сте о различитим деловима Флак већ апликације, а сада је време да напишете своју прву. Скрипта апликације `hello.py` приказана у [Примеру 2-1](#) дефинише инстанцу апликације и једну функцију руте и приказа, као што је раније описано.

Пример 2-1. `hello.py`: Комплетна Флак апликација

```
из flask импорт Flask app
= Flask(__name__)

@app.route('/')
def index():
    return 'Здраво свет!</x1>'
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, сада можете покренути гит цхецкоут 2а да исте проверили ову верзију апликације.

## Ве сервер за развој

Флакс апликације укључују развојни ве сервер који се може покренути командом флакс рун . Ова команда тражи име Питхон скрипте која садржи инстанцу апликације у променљивој окружењу ФЛАСК\_АПП .

Да исте покренули апликацију хелло.пи из претходног одељка, прво се уверите да је виртуелно окружење које сте креирали раније активирано и да је у њему инсталација Флакс. За кориснике Линук-а и мацОС-а, покрените ве сервер на следећи начин:

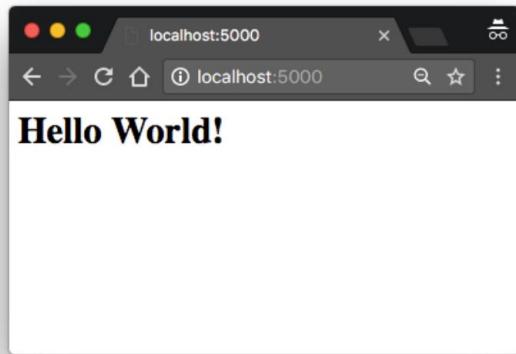
```
(венв) $ екпорт ФЛАСК_АПП=хелло.пи (венв)
$ флакс рун * Сервинг Флакс апликација
"здраво"
* Покреће се на http://127.0.0.1:5000/ (притисните ЦТРЛ+Ц да исте изашли)
```

За кориснике Микрософт Виндовс-а, једина разлика је у томе како је подешена променљива окружења ФЛАСК\_АПП :

```
(венв) $ сет ФЛАСК_АПП=хелло.пи (венв)
$ флакс рун * Сервинг Флакс апликација
"здраво"
* Покреће се на http://127.0.0.1:5000/ (притисните ЦТРЛ+Ц да исте изашли)
```

Када се сервер покрене, прелази у петљу која приhvата захтеве и сервисира их. Ова петља се наставља све док не зауставите апликацију притиском на Цтрл+Ц.

Док је сервер покренут, отворите свој ве претраживач и унесите http://лоџалхост:5000/ у траку за адресу. [Слика 2-1](#) показује шта ћете видети након повезивања са апликацијом.



Слика 2-1. хелло.пи Флакс апликација

Ако унесете ило шта друго после основног УРЛ-а, апликација неће знати како да се носи са тим и вратиће шифру грешке 404 у прегледач – позната грешка коју до ијате када се крећете до ве странице која не постоји.



Ве сервер који о ез еђује Флакс је намењен само за развој и тестирање. О производним ве серверима ћете научити у [17. поглављу](#).



Флакс развојни ве сервер се такође може покренути програмски позивањем методе `апп.рун()`. Старије верзије Флакс-а које нису имале команду `флакс` захтевале су да се сервер покрене покретањем главне скрипте апликације, која је морала да садржи следећи исечак на крају:

```
ако __наме__ == '__майн__':
    апп.рун()
```

Док команда `флакс рун` чини ову практику непотребном, метода `апп.рун()` и даље може исти корисна у одређеним приликама, као што је тестирање јединица, као што ћете научити у [поглављу 15.](#)

## Динамиц Роутес

Друга верзија апликације, приказана у [Примеру 2-2](#), додаје другу руту која је динамичка. Када посетите динамички УРЛ у свом претраживачу, иће вам представљен персонализовани поздрав који укључује име наведено у УРЛ-у.

Пример 2-2. хелло.пи: Флакс апликација са динамичком рутом

```
из flask импорт Flask апп
= Flask(__name__)

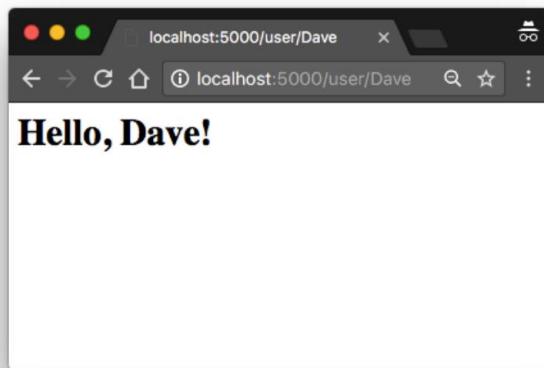
@app.route('/')
дeф индек( ):
    ретурн '<x1>Здраво свет!</x1>'

@app.route('/усер/<наме>')
дeф усер(наме): ретурн
    '<x1>Здраво, {1:</x1>}.формат(наме)
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, сада можете покренути гит цхеџкоут 2 да исте проверили ову верзију апликације.

Да исте тестирали динамичку руту, уверите се да сервер ради, а затим идите на `http://лоцалхост:5000/усер/Даве`. Апликација ће одговорити персонализованим поздравом користећи динамички аргумент имена . Покушајте да користите различита имена у УРЛ-у да видите како функција приказа увек генерише одговор на основу датог имена. Пример је приказан на [слици 2-2](#).



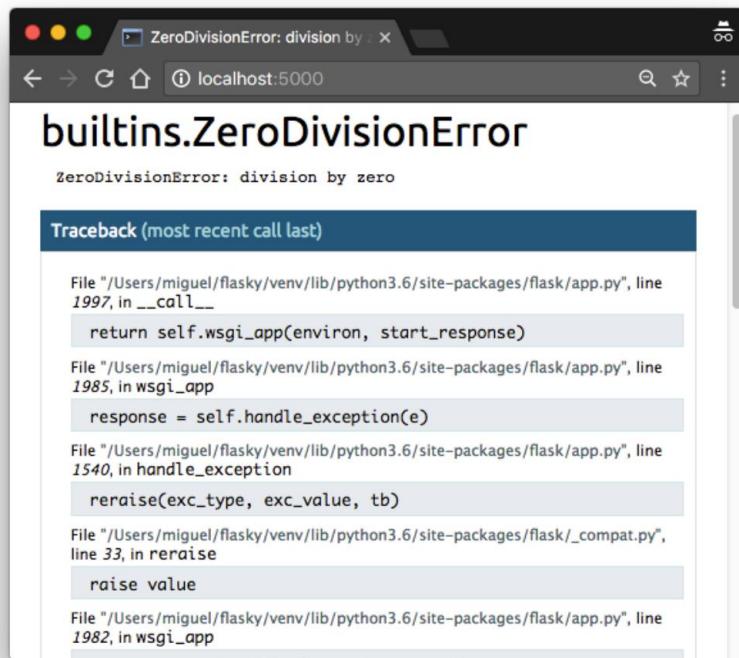
Слика 2-2. Динамичка пута

#### Режим за отклањање грешака

Флакс апликације могу опционо да се изврше у режиму за отклањање грешака. У овом режиму су подразумевано омогућена два веома згодна модула развојног сервера под називом релоадер и деаггер.

Када је поновно учитавање омогућено, Флакс прати све датотеке изворног кода вашег пројекта и аутоматски поново покреће сервер када се ило која од датотека измени. Имати сервер који ради са укљученим релоадером је изузетно корисно током развоја, јер сваки пут када измените и сачувате изворну датотеку, сервер се аутоматски поново покреће и преузима промену.

Деаггер је веб-азирана алатка која се појављује у вашем претраживачу када ваша апликација покрене неօрађени изузетак. Прозор већ претраживача се трансформише у интерактивни стек који вам омогућава да прегледате изворни код и процените изразе на ило ком мести у стеку позива. Можете видети како деаггер изгледа на [слици 2-3](#).



Слика 2-3. Флакс де уггер

Подразумевано, режим за отклањање грешака је онемогућен. Да иште то омогућили, поставите `ФЛАСК_ДЕБУГ=1` променљиву окружења пре него што позовете `флакс рун`:

```
(венв) $ екпорт ФЛАСК_АПП=хелло.пи (венв) $ екпорт
ФЛАСК_ДЕБУГ=1 (венв) $ флакс рун * Сервинг Флакс
апликација "здраво"

* Форсирање режима за отклањање
грешака * Покреће се на http://127.0.0.1:5000/ (притисните ЦТРЛ+Ц да иште изашли)
* Поновно покретање са
статистиком * Де агер је активан!
* ПИН за отклањање грешака: 273-181-528
```

Ако користите Microsoft Windows, користите сет уместо екпорт да иште подесили променљиве окружења.



Ако покренете сервер помоћу методе апп.рун(), променљиве окружења ФЛАСК\_АПП и ФЛАСК\_ДЕБУГ се не користе. Да исте програмски омогућили режим за отклањање грешака, користите апп.рун(де уг=Труе).



Никада не омогућавајте режим за отклањање грешака на производном серверу. Де агер посе но омогућава клијенту да захтева даљинско извршење кода, тако да ваш производни сервер чини ранјивим на нападе. Као једноставна заштитна мера, програм за отклањање грешака тре а да се активира помоћу ПИН-а, одштампаног на конзоли помоћу команде за покретање очице.

## Опције командне линије

Команда флакс подржава ројне опције. Да исте видели шта је доступно, можете покренути флакс --хелп или једноставно флакс ез икаквих аргумента:

```
(вени) $ флакс --хелп
```

Употребе а: флакс [ОПЦИЈЕ] КОМАНДА [АРГС]...

Ова наред а љуске делује као општа услужна скрипта за Флакс апликације.

Учитава конфигурисану апликацију (преко променљиве окружења ФЛАСК\_АПП), а затим о ез еђује команде које о ез еђује апликација или сам Флакс.

Најкорисније команде су наред е "покрени" и "љуска".

Пример употребе е:

```
$ екпорт ФЛАСК_АПП=хелло.пи $  
екпорт ФЛАСК_ДЕБУГ=1 $ флакс рун
```

Опције: -

верзион Прикажи верзију флакс-а --хелп

Прикажи ову поруку и изађи.

команде:

трдати Покреће развојни сервер. скелл

Покреће љуску у контексту апликације.

Команда флакс скелл се користи за покретање сесије Питхон љуске у контексту апликације. Ову сесију можете користити за покретање задатака одржавања или тестова или за отклањање грешака. Стварни примери где је ова команда корисна иће представљени касније, у неколико поглавља.

Већ сте упознати са командом флакс рун , која, као што јој назив говори, покреће апликацију са развојним ве сервером. Ова команда има много опција:

```
(венв) $ flask run --help
Употребе а: трчање у оци [ОПЦИЈЕ]
```

Покреће локални развојни сервер за Флакс апликацију.

Овај локални сервер се препоручује само за развојне сврхе, али се може користити и за једноставну интранет примену. Подразумевано неће подржавати ило какву врсту паралелности да и се поједноставило отклањање грешака. Ово се може променити опцијом --витх-тхреадс која ће омогућити основну вишеникту о раду.

Програм за поновно учитавање и отклањање грешака су подразумевано омогућени ако је ознака за отклањање грешака у Флакс-у омогућена или онемогућена у супротном.

Опције: -x,

--хост ТЕКСТ -п, --порт

Интерфејс за повезивање.

ИНТЕГР --релоад / --но-

Порт за повезивање.

релоад

Омогућите или онемогућите поновно учитавање. Подразумевано, програм за поновно учитавање је активан ако је омогућено отклањање грешака.

--де уггер / --но-де уггер

Омогућите или онемогућите програм за отклањање грешака. Подразумевано је програм за отклањање грешака активан ако је омогућено отклањање грешака.

--еагер-лоадинг / --лази-лоаддер

Омогућите или онемогућите жељно учитавање. Подразумевано је жељно учитавање омогућено ако је релоадер онемогућен. -- витх-тхреадс / --вихтх-тхреадс

-помоћ

Омогућите или онемогућите вишеникту.

Прикажите ову поруку и изађите.

Аргумент --хост је посе но користан јер говори ве серверу који мрежни интерфејс тре да слуша за везе од клијената. Подразумевано, Флаксов развојни ве сервер слуша везе на локалном хосту, тако да се прихватавају само везе које потичу са рачунара који покреће сервер. Следећа команда чини да ве сервер слуша везе на интерфејсу јавне мреже, омогућавајући и другим рачунарима у истој мрежи да се повежу:

```
(венв) $ flask run --host 0.0.0.0 * Сервинг Флакс
апликација "здраво"
* Покреће се на http://0.0.0.0:5000/ (притисните ЦТРЛ+Ц да исте изашли)
```

Ве сервер и сада тре ало да уде доступан са ило ког рачунара у мрежи на адреси хттп:// а цд:5000, где је а цд ИП адреса рачунара који покреће сервер у вашој мрежи.

Опције --релоад, --но -релоад, --де уггер и --но -де уггер пружају већи степен контроле на врху подешавања режима за отклањање грешака. На пример, ако де уг

режим је омогућен, --но-де уггер се може користити за искључивање програма за отклањање грешака, док је режим за отклањање грешака и програм за поновно учитавање омогућени.

## Циклус захтев-одговор

Сада када сте се играли са основном апликацијом Фласк, можда исте желели да сазнате више о томе како Фласк ради своју магију. Следећи одељци описују неке од аспеката дизајна оквира.

### Контексти апликације и захтева Када Фласк

прими захтев од клијента, трећа да учини неколико операција најеката доступним функцији приказа која ће њиме управљати. До тада пример је овакат захтева, који инкапсулира ХТТП захтев који шаље клијент.

Очигледан начин на који и Фласк могао дати функцији погледа приступ овакту захтева је слање као аргумент, али то и захтевало да свака појединачна функција погледа у апликацији има додатни аргумент. Ствари постају компликованије ако узмете у обзир да овакат захтева није једини овакат коме функције прегледа можда треба да приступе да испуниле захтев.

Да и избегао затрпавање функција приказа са пуно аргумента који можда нису увек потребни, Фласк користи контексте да привремено учини одређене операције глобално доступним. Захваљујући контекстима, функције приказа као што је ова могу се написати:

[из захтева за увоз алона](#)

```
@апп.роуте('/')
дeф индeк():
    усер_агент = рекуест.хeадерс.гет ('Усер-Агент') рeтурн
    '<п>Ваш прeглeдaч je {}</п>'.формат(усер_агент)
```

Овдјелате пажњу на то како се у овој функцији приказа захтев користи као да је глобална променљива. У стварности, захтев не може ити глобална варијабла; у вишенинтном серверу неколико нити може радити на различитим захтевима од различитих клијената све у исто време, тако да свака нит мора да види другачији овакат у захтеву. Контексти омогућавају Фласку да одређене варијабле учини глобално доступним нити изометања других нити.



Нит је најмањи низ инструкција којим се може управљати независно. Уочено је да процес има више активних нити, понекад деле ресурсе као што су меморија или ручке датотека. Вишенинтни веб сервери покрећу скуп нити иирају нит из скупа за оваког долазног захтева.

У Фласку постоје два контекста: контекст апликације и контекст захтева.

Та ела 2-1 приказује варија ле изложене у сваком од ових контекста.

Та ела 2-1. Гло ални контекст фласк

Име променљиве	Контекст	Опис
цуррент_апп	Контекст апликације	Инстанца апликације за активну апликацију.
Г	Апликација контекст	Ојекат који апликација може да користи за привремено складиштење током ораде захтева. Ова променљива се ресетује са сваким захтевом.
захтев	Контекст захтева	Ојекат захтева, који инкапсулира садржај ХТТП захтева који је послao клијент.
седница	Контекст захтева	Корисничка сесија, речник који апликација може да користи за чување вредности које се „памте“ између захтева.

Фласк активира (или гура) контексте апликације и захтева пре слања захтева апликацији и уклања их након што се захтев оради. Када се контекст апликације гурне, променљиве цуррент\_апп и г постају доступне нити. Слично томе, када је контекст захтева гурнут, захтев и сесија такође постају доступни. Ако се ило којој од ових променљивих приступи из активне апликације или контекста захтева, генерише се грешка. Четири контекстуалне варија ле ће ити детаљно орађене у овом и каснијим поглављима, тако да не прините ако још увек не разумете зашто су корисне.

Следећа сесија Питхон љуске показује како функционише контекст апликације:

```
>>> из апликације хелло
импорт >>> из фласк импорт
цуррент_апп >>> цуррент_апп.наме
Траце ацк (последњи позив последњи):
...
РунтимеError: ради ван контекста апликације >>> апп_цтк
= апп.апп_цонтект() >>> апп_цтк.пушч() >>> цуррент_апп.наме
'здраво' >>> апп_цтк.поп()
```

У овом примеру, цуррент\_апп.наме не успева када није активан контекст апликације, али постаје важећи када се контекст апликације за апликацију гурне. Оратите пажњу на то како се контекст апликације доја позивањем апп.апп\_цонтект() на инстанци апликације.

Отпремање захтева Када

апликација прими захтев од клијента, тре да сазна коју функцију прегледа да позове да и је сервисирала. За овај задатак, Фласк тражи УРЛ дату у

захтева у УРЛ мапи апликације, која садржи мапирање УРЛ-ова у функције приказа које њима рукују. Флакс прави ову мапу користећи податке дате у декоратору апп.роуте , или еквивалентној верзији `ез` декоратора, апп.адд\_урл\_руле().

Да исте видели како изгледа УРЛ мапа у Флакс апликацији, можете прегледати мапу креирану за хелло.пи у Питхон лъсци. Пре него што ово покушате, уверите се да је ваше виртуелно окружење активирано:

```
(венв) $ питхон >>> из
апликације хелло импорт >>> апп.урл_мап
Мап([<Правило '/ (ХЕАД, ОПТИОНС, ГЕТ) ->
индеク>, <Правило '/статиц/<филенаме>' (ХЕАД, ОПТИОНС, ГЕТ) -> статиц>,
<Правило '/усер/<наме>' (ХЕАД, ОПТИОНС, ГЕТ) -> корисник>])
```

Руте / и /усер/<наме> су дефинисали декоратори апп.роуте у апликацији. /статиц/ <филенаме> ruta је посе на ruta коју је додао Флакс да и се омогућио приступ статичним датотекама. Више о статичким датотекама сазнаћете у 3. [поглављу](#).

Елементи (ХЕАД, ОПТИОНС, ГЕТ) приказани у УРЛ мапи су методе захтева којима руте рукују. ХТТП спецификација дефинише да се сви захтеви издају са методом, која оично показује коју радњу клијент тражи од сервера да изврши. Флакс прилаже методе за сваку руту тако да различите методе захтева послате на исту УРЛ адресу могу да се орађују помоћу различитих функција приказа. Флакс аутоматски управља методама ХЕАД и ОПТИОНС , тако да се у пракси може рећи да су у овој апликацији три руте у УРЛ мапи придружене ГЕТ методи, која се користи када клијент жели да затражи информације као што је већа страна. Научићете како да креirate руте за друге методе захтева у 4. [поглављу](#) .

## О јекат захтева Видели

сте да Флакс излаже о јекат захтева као контекстуалну променљиву под називом захтев. Ово је изузетно користан о јекат који садржи све информације које је клијент укључио у ХТТП захтев. [Та ела 2-2](#) најчешће коришћене атријуте и методе о јекта захтева Флакс.

Та ела 2-2. Флакс рекуест о јеџт

Опис атријута или методе	
форму	Речник са свим пољима о расца достављеним уз захтев.
аргс	Речник са свим аргументима прослеђеним у стрингу упита УРЛ-а.
вредности	Речник који комбинује вредности у форми и аргументима.
колачићи	Речник са свим колачићима укљученим у захтев.
заглавља	Речник са свим ХТТП заглављима укљученим у захтев.
фајлови	Речник са свим отпремљеним датотекама укљученим у захтев.

## Опис атријута или методе

гет_дата()	Враћа афероване податке из тела захтева.
гет_јсон()	Враћа Питхон речник са рашчлањеним ЈСОН-ом укљученим у тело захтева.
нацрт	Име Флакс нацрта који орађује захтев. Нацрти су представљени у <a href="#">поглављу 7</a> .
крајња тачка	Име Флакс крајње тачке која орађује захтев. Флакс користи име функције приказа као назив крајње тачке за руту.
методом	Метод ХТТП захтева, као што је ГЕТ или ПОСТ.
шема	Шема УРЛ адресе (хттп или хттпс).
ис_сецуре()	Враћа Тачно ако је захтев дошао преко езедне (ХТТПС) везе.
домаћин	Хост дефинисан у захтеву, укључујући број порта ако га је дао клијент.
пут	Део путање УРЛ-а.
куери_стринг	Део стринга упита УРЛ адресе, као нео рађена инарна вредност.
пуна путања	Делови УРЛ-а са путањом и стрингом упита.
урл	Комплетна УРЛ адреса коју захтева клијент.
асе_урл	Исто као и урл, али ез компоненте стринга упита.
ремоте_аддр	ИП адреса клијента.
околина	Нео рађени речник ВСГИ окружења за захтев.

## Рекуест Хоокс

Понекад је корисно извршити код пре или после ораде сваког захтева. За пример, на почетку сваког захтева можда ће ити потре но да се креира веза са азом података. или аутентификовати корисника који подноси захтев. Уместо да дуплира код који оавља ове радње у свакој функцији приказа, Флакс вам даје опцију да се региструјете уо ичайене функције које се позивају пре или после слања захтева.

Куке за захтеве су имплементиране као декоратори. Ово су четири куке које подржавају флаша:

### пре\_захтева

Региструје функцију за покретање пре сваког захтева.

### пре\_првог\_захтева

Региструје функцију за покретање само пре него што се оради први захтев. Ово може ити а погодан начин за додавање задатака иницијализације сервера.

### афтер\_рекуест

Региструје функцију за покретање након сваког захтева, али само ако нема нео рађених изузетака. десиле су се ције.

### теардовн\_рекуест

Региструје функцију која се покреће након сваког захтева, чак и ако се не орађују изузети дошло.

Уо и чајени о разац за дељење података између функција закачивања захтева и функција прегледа је коришћење глобалног контекста као складиштења. На пример, овај ривичар ефоре\_рекуест може учитати пријављеног корисника из азе података и ускладишити га у гусер. Касније, када се позове функција приказа, она може да преузме корисника одатле.

Примери закачивачи захтева и ће приказани у наредним поглављима, тако да не прините ако намена ових кукица још увек нема смисла.

### Одговори Када

Флакс позове функцију приказа, очекује да њена повратна вредност буде одговор на захтев. У већини случајева одговор је једноставан стринг који се шаље назад клијенту као ХТМЛ страница.

Али ХТТП протокол захтева више од стринга као одговор на захтев. Веома важан део ХТТП одговора је статусни код, који Флакс подразумевано поставља на 200, код који указује да је захтев успешно обavljen.

Када функција приказа треба да одговори другим статусним кодом, може додати нумерички код као другу повратну вредност после текста одговора. На пример, следећа функција приказа враћа статусни код 400, код за грешку лош захтев:

```
@апп.роуте('/')
деф индек():
    враћа '<x1>Лош захтев</x1>', 400
```

Одговори које враћају функције приказа могу такође узети трећи аргумент, речник заглавља која се додају у ХТТП одговор. Видете пример прилагођених заглавља одговора у [поглављу 14](#).

Уместо да враћају једну, две или три вредности као тупле, функције Флакс виев имају опцију враћања јекта одговора. Функција маке\_респонсе() узима један, два или три аргумента, исте вредности које се могу вратити из функције приказа, и враћа еквивалентни јекат одговора. Понекад је корисно генерисати јекат одговора унутар функције приказа, а затим користити његове методе за даље конфигурисање одговора. Следећи пример креира јекат одговора и затим поставља колачић у њему:

```
из flask импорт маке_респонсе

@апп.роуте('/')
деф индек():
    респонсе = маке_респонсе('<x1>Овај документ носи колачић!</x1>')
    респонсе.сет_цоокие('ансвер', '42') врати одговор
```

Тај ела 2-3 приказује најчешће коришћене атријуте и методе доступне у јектима одговора.

Та је 2-3. О јекат одговора тиквице

Опис атријута или методе	
статус_код	Нумерички ХТТП статусни код
заглавља	О јекат сличан речнику са свим заглављима која ће ити послата са одговором
сет_цоокије()	Додаје колачић одговору
делете_цоокије()	Уклања колачић
контент_ленгтх	Дужина тела одговора
Тип садржаја	Тип медија тела одговора
сет_дата()	Поставља тело одговора као вредност низа или ајтова
гет_дата()	До ија тело одговора

Постоји посећа на врста одговора која се зове преусмеравање. Овај одговор не укључује документ странице; само даје претраживачу нову УРЛ адресу до које се може кретати. Веома јо ичајена употреба преусмеравања је када радите са већим расцима, као што ћете научити у [поглављу 4](#).

Преусмеравање је описано иначе назначено кодом статуса одговора 302 и УРЛ-ом на коју треба да идете датим у заглављу локације. Одговор за преусмеравање се може генерирати ручно са повратком од три вредности или са објектом одговора, или са око зиром на његову честу употребу, у Фласку ове вредности еђује помоћну функцију `редирект()` која креира овај тип одговора:

[са преусмеравања увоза фласк](#)

```
@апп.роуте('/') деф
индек( ):
    ретурн редирект('хттп://ввв.екампле.цом')
```

Још један посебан одговор се издаје са функцијом `орт()`, која се користи за руковање грешкама. Следећи пример враћа статусни код 404 ако динамички аргумент ид дат у УРЛ-у не представља важећег корисника:

[из фласк увоз прекинут](#)

```
@апп.роуте('/усер/<ид>') деф
гет_усер(ид): усер =
    лоад_усер(ид) ако није
        усер: а орт(404) ретурн
            '<x1>Здраво, {}<
            x1>'.формат(корисничко име)
```

Имајте на уму да ако `орт()` не враћа контролу назад у функцију јер покреће изузетак.

## Фласк Ектенсионс

Боца је дизајнирана да се продужи. Намерно се задржава ван о ласти важне функционалности као што су аза података и аутентификација корисника, дајући вам сло оду да иза ерете пакете који нај оље одговарају вашој апликацији или да напишете своје ако желите.

Заједница је креирала велики из оп Фласк ектензија за различите сврхе, а ако то није довољно, може се користити и ило који стандардни Питхон пакет или и лиотека. Свој први Фласк ектензију ћете користити у [Поглављу 3](#).

Ово поглавље уводи концепт одговора на захтеве, али има још много тога да се каже о одговорима. Фласк пружа веома до ру подршку за генерисање одговора помоћу ша лона, а ово је толико важна тема да је следеће поглавље посвећено њој.

## ПОГЛАВЉЕ 3

## Ша лони

Кључ за писање апликација које се лако одржавају је писање чистог и до ро структурираног кода. Примери које сте до сада видели су сувише једноставни да и то демонстрирали, али функције Флакс виев имају две потпуно независне сврхе прерушене у једну, што ствара про лем.

Очигледни задатак функције приказа је да генерише одговор на захтев, као што сте видели у примерима приказаним у [Поглављу 2](#). За најједноставније захтеве ово је доволно, али у многим случајевима захтев такође покреће промену стања апликација, а функција приказа је место где се ова промена генерише.

На пример, узмите у о зир корисника који региструје нови налог на ве локацији. Корисник уписује адресу е-поште и лозинку у ве о разац и кликне на дугме Пошаљи. На сервер стиже захтев са подацима које је дао корисник, а Флакс га шаље функцији приказа која о рађује захтеве за регистрацију. Ова функција прегледа тре а да разговара са азом података да и додала новог корисника, а затим генерисала одговор за слање назад претраживачу који укључује поруку о успеху или неуспеху. Ове две врсте задатака се формално називају пословна логика и логика презентације, респективно.

Мешање пословне и презентацијске логике доводи до кода који је тешко разумети и одржавати. Замислите да морате да направите ХТМЛ код за велику та елу спајањем података до ијених из азе података са неопходним литералима ХТМЛ стрингова. Премештање логике презентације у ша лоне помаже у по ољшању могућности одржавања апликације.

Ша лон је датотека која садржи текст одговора, са променљивим чвара места за динамичке делове који ће ити познати само у контексту захтева. Процес који замењује променљиве стварним вредностима и враћа коначан низ одговора назива се рендеровање. За задатак рендеровања ша лона, Флакс користи моћни механизам ша лона који се зове Јиња2.

## Тхе Јиња2 Темплате Енгине

У свом најједноставнијем облику, Јиња2 шаблон је датотека која садржи текст одговора.

Пример 3-1 приказује шаблон Јиња2 који одговара одговору функције приказа индекса из Примера 2-1.

Пример 3-1. темплатес/индек.хтмл: шаблон Јиња2

```
<x1>Здраво свет!</x1>
```

Одговор који враћа функција приказа усер() из Примера 2-2 има динамичку компоненту, која је представљена променљивом. Пример 3-2 показује шаблон који имплементира овај одговор.

Пример 3-2. темплатес/усер.хтмл: шаблон Јиња2

```
<x1>Здраво, {{ name }}!</x1>
```

Рендеринг Темплатес

Стандардно Флакс тражи шаблоне у поддиректоријуму шаблона који се налази унутар главног директоријума апликације. За следећу верзију хелло.пи, потребно је да креирате поддиректоријум шаблона и да у њега складиштите шаблоне дефинисане у претходним примерима као индек.хтмл и усер.хтмл, респективно.

Функције приказа у апликацији морају бити модификоване да биду и се приказали ови шаблони. Пример 3-3 показује ове промене.

Пример 3-3. хелло.пи: приказивање шаблона

```
из flask импорт Flask, render_template

# ...

@app.route('/')
дeф индекс():
    рeтурн render_template('индек.хтмл')

@app.route('/усер/<имe>') дeф
    усер(имe): рeтурн
        render_template('усер.хтмл', имe=имe)
```

Функција `рендер_темплате()` коју о `ез` еђује Флакс интегрише Јиња2 ша лонски механизам са апликацијом. Ова функција узима име датотеке ша лона као први аргумент. Сви додатни аргументи су парови кључ-вредност који представљају стварне вредности за променљиве на које се упућује у ша лону. У овом примеру, други ша лон прима променљиву имена .

Аргументи кључних речи као што је `име=име` у претходном примеру су прилично чести, али могу изгледати з `уњујуће` и тешко их је разумети ако нисте навикили на њих. „Име“ на левој страни представља име аргумента, које се користи у чувару места написаном у ша лону. „Име“ на десној страни је променљива у тренутном опсегу која о `ез` еђује вредност за аргумент истог имена. Иако је ово уо ичайен о разац, коришћење истог имена променљиве са о `е` стране није потре но.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкуот За да исте проверили ову верзију апликације.

## Променљиве

Конструкција `{{ наме }}` која се користи у ша лону приказаном у [Примеру 3-2](#) упућује на променљиву, посе ан чувар места који говори механизму ша лона да вредност која иде на то место тре а да се до ије из података који су дати у тренутку када је ша лон рен - деред.

Јиња2 препознаје променљиве ило ког типа, чак и сложене типове као што су листе, речници и о јекти. Следи још неколико примера варија ли које се користе у ша лонима:

```
<п> Вредност из речника: {{ мидицт['кеи'] }}.</п> <п> Вредност из листе: {{ милист[3] }}.</п> <п> А вредност са листе, са променљивим индексом: {{ милист[минтвар] }}.</п> <п> Вредност из методе о јекта: {{ мио .ј.сомеметод() }}.</п>
```

Променљиве се могу модификовати помоћу лтера, који се додају иза имена променљиве са знаком црне линије као сепаратором. На пример, следећи ша лон приказује име променљиве великим словима:

Здраво, {{ име |велика слова }}

Та ела [3-1](#) наводи неке од најчешће коришћених филтера који долазе са Јиња2.

Та је 3-1. Јиња2 варија илни литри

Назив филтера	Опис
ез едно	Рендерује вредност ез примене из егавања
капитализе	Претвара први знак вредности у велика слова, а остатак у мала слова
ниже	Конвертује вредност у мала слова
горњи	Конвертује вредност у велика слова
наслов	Сваку реч у вредности пише великим словом
трим	Уклања почетне и задње размаке из трака
вредности	Уклања све ХТМЛ ознаке из вредности пре приказивања

Занимљиво је истакнути сигуран филтер. Подразумевано, Јиња2 из егава све варија ле из ез једносних разлога. На пример, ако је променљива подешена на вредност '<x1>Здраво</x1>', Јиња2 ће приказати стринг као '&lt;x1&gt;Здраво&lt;/x1&gt;', што ће довести до приказа елемента x1 и не тумачи претраживач. Много пута је потребно приказати ХТМЛ код усклађиштен у променљивим, а за те случајеве се користи сигуран филтер.



Никада не користите ез један филтер за вредности које нису поуздане, као што је текст који су корисници унели у веб расце.

Комплетна листа филтера може се доћи из званичне [Јиња2 документације](#).

## Контролне структуре

Јиња2 нуди неколико контролних структура које се могу користити за промену тока шаблона. Овај одељак уводи неке од најкориснијих са једноставним примерима.

Следећи пример показује како се условни искази могу унети у шаблон:

```
{% иф user %}
    Здраво, {{ корисник }}!
{% елсе %}
    Здраво странче! %
    ендиф %}
```

Још једна уочијена потреба је да у шаблонима је да се прикаже листа елемената. Овај пример показује како се то може урадити са фор петљом:

```
<ул>
    {% за коментар у коментарима %}
```

```
<ли>{{ цоммент }}</ли> {%
    ендфор %} </ул>
```

Јиња2 такође подржава макрое, који су слични функцијама у Питхон коду. На пример:

```
{% мацро рендер_цоммент(цоммент) %}
<ли>{{ цоммент }}</ли> {%
    ендмацро
    %}

<ул>
{%
    за коментар у коментарима %
        {{ рендер_цоммент(цоммент) }} {%
            ендфор %} </ул>
```

Да и се макрои учинили више употребе љивим, могу се чувати у самосталним датотекама које се затим увозе из свих ша лона којима су потре ни:

```
{% импорт 'мацрос.хтмл' као макроа %}
<ул> {%
    за коментар у коментарима %
        {{ мацрос.рендер_цоммент(цоммент) }}

    {% ендфор %}
</ул>
```

Делови кода ша лона који тре да се понављају на неколико места могу се сачувати у засе ној датотеци и укључити из свих ша лона како и се из егло понављање:

```
{% инклуде 'цоммон.хтмл' %}
```

Још један моћан начин поновне употребе је наслеђивање ша лона, које је слично наслеђивању класа у Питхон коду. Прво се креира основни ша лон са именом асе.хтмл:

```
<хтмл>
<хеад>
    {% лоцк хеад %}
    <титле>{% лоцк титле %}{% енд лоцк %} - Моја апликација</титле> {%
        енд лоцк %} <хеад> < оди> {% лоцк оди %} {% енд лоцк %} </ оди>
</хтмл>
```

Основни ша лони дефинишу локове који се могу заменити изведенним ша лонима. Џиња2 лоцк и енд лоцк директиве дефинишу локове садржаја који се додају ази ша лон. У овом примеру постоје локови који се зову глава, наслов и тело; имајте на уму да је наслов садржан у глави. Следећи пример је изведен ша лон основног ша лона:

```
{% проширује " асе.хтмл" %}
{%
    лоцк титле %}Индекс{%
    енд лоцк %}
{%
    глава лока %} {{ супер() }} <стиле> </
стиле> {% енд лоцк %} {% лок оди
%} <x1>Здраво, свет!</x1> {% енд лоцк
%}
```

Директива ектендс изјављује да овај ша лон потиче од асе.хтмл. Ову директиву прате нове дефиниције за три лока дефинисана у основном ша лону, који су уметнути на одговарајућа места. Када лок има неки садржај иу основном иу изведеном ша лону, користи се садржај из изведеног ша лона. Унутар овог лока, изведени ша лон може позвати супер() да референцира садржај лока у основном ша лону. У претходном примеру, ово је урађено у локу главе .

Стварна употребе а свих контролних структура представљених у овом одељку иће приказана касније, тако да ћете имати прилику да видите како функционишу.

## Боотстрап интеграција са Флакс-Боотстррап-ом

**Боотстрап** је оквир ве претраживача отвореног кода из Твитера који о ез еђује компоненте корисничког интерфејса које помажу у креирању чистих и атрактивних ве страница које су компати илне са свим модерним ве претраживачима који се користе на десктоп и мобилним платформама.

Боотстрап је оквир на страни клијента, тако да сервер није директно укључен у њега. Све што сервер тре а да уради је да о ез еди ХТМЛ одговоре који упућују на Боотстрап-ове каскадне та еле стилова (ЦСС) и ЈаваСкрипт датотеке, и инстанцирају жељене елементе корисничког интерфејса кроз ХТМЛ, ЦСС и ЈаваСкрипт код. Идеално место за све ово је у ша лонима.

Наиван приступ интеграцији Боотстрап-а са апликацијом је да се унесе све неопходне промене у ХТМЛ ша лоне, пратећи препоруке дате у документацији Боотстрата. Али ово је о ласт у којој употребе а Флакс екстензије чини задатак интеграције много једноставнијим, док помаже да ове промене уду лепо организоване.

Екстензија се зове Флакс-Боотстрап и може се инсталирати помоћу пип-а:

(веб) \$ пип инсталл флакс- боотстрап

Екстензије флакс-а се иницијализују исто време када се креира инстанца апликације.

**Пример 3-4** показује иницијализацију Флакс-Боотстрап-а.

### Пример 3-4. хелло.пи: Иницијализација Фласк-Боотстрап-а

```
фром flask_ _оотстрап импорт Боотстррап #
оотстррап = Боотстррап(апп)
```

Екстензија се о ично увози из `фласк_<име>` пакета, где је `<име>` име екстензије. Већина Фласк екстензија прати један од два конзистентна о расца за иницијализацију. У [примеру 3-4](#), екстензија се иницијализује преношењем инстанце апликације као аргумента у конструктору. Научићете о напреднијој методи за иницијализацију екстензија прикладних за веће апликације у 7. [поглављу](#).

Када се Фласк-Боотстрап иницијализује, основни ша лон који укључује све Боотстррап датотеке и општу структуру је доступан апликацији. Апликација затим користи предности наслеђа ша лона Јиња2 да прошири овај основни ша лон. [Пример 3-5](#) приказује нову верзију усер.хтмл као изведени ша лон.

### Пример 3-5. темплатес/усер.хтмл: ша лон који користи Фласк-Боотстррап

```
{% проширује " оотстрап/ асе.хтмл" %}

{% лоцк титле %}Фласки{% енд лоцк %}

{% лоцк нав ар %}

<див цласс="нав ар нав ар-инверсе" роле="навигатион"> <див
    цласс="цонтайнер"> <див цласс="нав ар-хеадер">

        < уттон типе=" уттон" цласс="нав ар-тоггле" data-
            тоггле="цоллapse" data-таргет=".нав ар-цоллapse"> <спан цласс="ср-
            онли">Укључи/искључи навигацију</спан> <спан цласс="ицион- ар"></
            спан> <спан цласс="ицион- ар"></спан> <спан цласс="ицион- ар"></
            спан> </ уттон> <a цласс="нав ар - ранд" хреф="/">Фласки </а> </
            див> <див цласс="нав ар-цоллapse цоллapse">

                <ул цласс="нав нав ар-нав">
                    <ли>< а хреф="/">Почетна</а></ли> </
                    ул> </див> </див> </див> {% енд лоцк %}

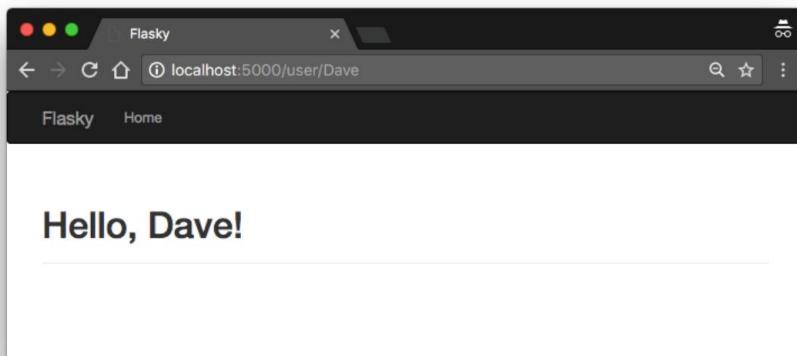
    {% лоцк цонтент %}
    <див цласс="цонтайнер"> <див
        цласс="паге-хеадер"> <x1>Здраво,
        {{ наме }}</x1> </див>
```

```
</див>
{% енд лоцк %}
```

Директива Јиња2 ектенд имплементира наслеђивање ша лона упућивањем на оотстрап/ асе.хтмл из Флакс-Боотстрап-а. Основни ша лон из Флакс-Боотстрап-а пружа скелет ве странице која укључује све БоотстрраЖСС и ЈаваСкрипт датотеке.

Ша лон усер.хтмл дефинише три лока под називом наслов, трака за навигацију и садржај. Ово су сви локови које основни ша лон извози за дефинисање изведеног ша лона. Насловни лок је једноставан; његов садржај ће се појавити између ознака <титле> у заглавље приказаног ХТМЛ документа. Навар и локови садржаја резервисани су за траку за навигацију страница и главни садржај.

У овом ша лону, лок за навигацију дефинише једноставну траку за навигацију користећи БоотстрраЖ компоненте. Блок садржаја има контејнер <див> са заглављем странице унутра. Поздравна линија која је ила у претходној верзији ша лона је сада унутар заглавља странице. [Слика 3-1](#) показује како апликација изгледа са овим променама.



Слика 3-1. Боотстрап ша лони



Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит цхеџоут З да исте проверили ову верзију апликације. Флакс-Боотстрап пакет такође трећа да је инсталiran у вашем виртуелном окружењу. Званична [документација БоотстрараЖ](#) је одличан ресурс за учење препун примера спремних за копирање/пест.

Флакс-БоотстрраЖ-ов асе.хтмл ша лон дефинише неколико других локова који се могу користити у изведеном ша лонима. [Та ела 3-2](#) приказује комплетну листу доступних локова.

Та ела 3-2. Блокови основног ша лона Флакс-Боотстрата

Назив лока	Опис
доц	Цео ХТМЛ документ
хтмл_аттри	с Атри ути унутар ознаке <хтмл>
хтмл	Садржај ознаке <хтмл>
глава	Садржај ознаке <хеад>
наслов	Садржај ознаке <титле>
метас	Листа <мета> ознака
стилова	ЦСС дефиниције
оди_аттри	с Атри ути унутар ознаке < оди>
тело	Садржај ознаке < оди>
нав ар	Кориснички дефинисана трака за навигацију
садржаја	Кориснички дефинисан садржај странице
скрипте	ЈаваСкрипт декларације на дну документа

Многе локове у **та ели 3-2** користи сам Флакс-Боотстрап, тако да их зао илази директно и изазвало про леме. На пример, локови стилова и скрипти су где су декларисане БоотстрраШ ЦСС и ЈаваСкрипт датотеке. Ако апликација тре а додајте свој садржај у лок који већ има неки садржај, а затим Јиња2-ов супер() функција мора да се користи. На пример, овако и тре ало да уде лок скрипти написано у изведеном ша лону за додавање нове ЈаваСкрипт датотеке у документ:

```
{% лок скрипти %}
{{ супер() }}
<скрипт типе="текст/јаваскрипт" срц="ми-скрипт.јс"></скрипт>
{% енд лоцк %}
```

### Странице прилагођених грешака

Када унесете неважећу руту у адресну траку претраживача, до ићете код 404 страница са грешком. У поређењу са страницама које покреће БоотстрраШ, подразумевана страница са грешком је сада превише о ичан и непривлачен, и нема конзистентност са стварним генерисаним страницама по апликацији.

Флакс омогућава апликацији да дефинише прилагођене странице са грешкама које се могу заснивати на тем- плоче, као о ичне руте. Две најчешће шифре грешке су 404, које се покрећу када клијент захтева страницу или руту која није позната, и 500, која се покреће када постоји нео рађен изузетак у апликацији. **Пример 3-6** показује како о ез едити цус- том руковоацима за ове две грешке помоћу декоратора апп.еррорхандлер .

Пример 3-6. хелло.пи: прилагођене странице са грешком

```
@апп.еррорхандлер(404)
деф паге_нот_фоунд(е):
    ретурн рендер_темплате('404.хтмл'), 404
```

```
@апп.еррорхандлер(500)
деф интернал_сервер_еррор(е):
    ретурн рендер_темплате('500.хтмл'), 500
```

Руковаоци грешкама враћају одговор, попут функција прегледа, али такође морају да врате нумерички статусни код који одговара грешци, што Флакс прикладно прихвата као другу повратну вредност.

Ша лоне на које се упућује у о рађивачима грешака морају ити написани. Ови ша лони и тре ало да имају исти изглед као и о ичне странице, тако да ће у овом случају имати траку за навигацију и заглавље странице које приказује поруку о грешци.

Једноставан начин за писање ових ша лона је да копирате темплатес/усер.хтмл у темплатес/404.хтмл и темплатес/500.хтмл, а затим промените елементе заглавља странице у ове две нове датотеке у одговарајуће поруке о грешци, али то ће генерисати много дуплирања.

Наслеђивање ша лона Јиња2 може помоћи у томе. На исти начин на који Флакс-Боотстррап о ез еђује основни ша лон са основним изгледом странице, апликација може да дефинише сопствени основни ша лон са униформним изгледом странице који укључује траку за навигацију и оставља садржај странице да се дефинише у изведеним ша лонима. . Пример 3-7 приказује ша лоне/ асе.хтмл, нови ша лон који наслеђује оотстрап/ асе.хтмл и дефинише траку за навигацију, али је сам по се и основни ша лон другог нивоа за друге ша лоне као што су темплатес/усер.хтмл, ша лони /404.хтмл и темплатес/500.хтмл.

Пример 3-7. темплатес/ асе.хтмл: основни ша лон апликације са траком за навигацију

```
{% проширује " оотстрап/ асе.хтмл" %}

{% лоцк титле %}Флакски{% енд лоцк %}

{% лоцк нав ар
%} <див класс="нав ар нав ар-инверсе" роле="навигацијон">
    <див класс="цонтаинер"> <див класс="нав ар-хеадер">

        < уттон типе=" уттон" класс="нав ар-тоггле"
            дата-тоггле="цоллапсе" дата-таргет=".нав ар-цоллапсе"> <спан
                класс="ср- онли">Укључи/искључи навигацију</спан> <спан
                класс="ицон- ар"></спан> <спан класс="ицон- ар"></
                спан> <спан класс="ицон- ар"></спан> </ уттон> <a
                класс="нав ар - ранд" хреф="/">Флакски</а>
```

```

</див>
<див цласс="нав ар-цоллапсе цоллапсе">
    <ул цласс="нав нав ар-нав">
        <ли>< а хреф="/">Почетна</а></ли> </
    ул> </див> </див> </див> {%
        енд лоцк %}

{%
    лоцк цонтент %
}
<див цласс="цонтаинер"> {%
    лоцк паге_цонтент %} {%
    енд лоцк %} </див> {%
    енд лоцк %}

```

Блок садржаја овог шаблона је само елемент контејнера `<див>` који означаваја нови празан блок под називом `паге_цонтент`, који изведенни шаблони могу да дефинишу.

Шаблони апликације ће сада наследити овај шаблон уместо директно од Фласк-Боотстреп-а. [Пример 3-8](#) показује колико је једноставно конструисати прилагођену страницу грешке кода 404 која наслеђује темплатес/ `ace.html`. Страница за грешку 500 је слична и можете је пронаћи у ГитХу спремишту за апликацију.

Пример 3-8. темплатес/404.html: прилагођена страница са грешком кода 404 која користи наслеђивање шаблона

```

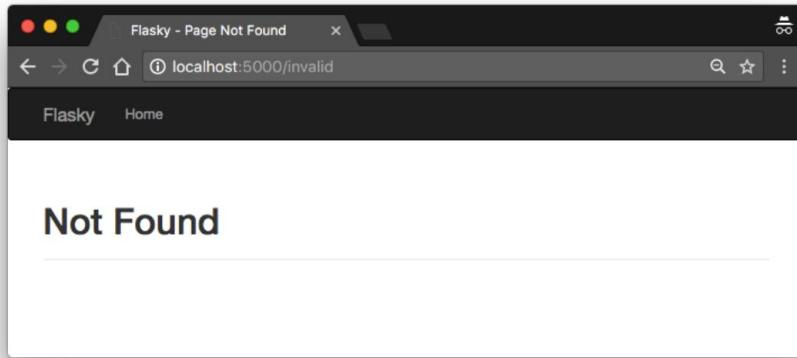
{%
    проширује " ace.html" %

{%
    лоцк титле %}Фласки – Страница није пронађена{%
    енд лоцк %}

{%
    лоцк паге_цонтент %} <див
    цласс="паге-хеадер" <x1>Није
    пронађено</x1> </див> {%
    енд лоцк %}

```

[Слика 3-2](#) приказује како страница са грешком изгледа у претраживачу.



Слика 3-2. Страница грешке прилагођеног кода 404

Ша лон темплатес/усер.хтмл сада се може поједноставити тако што ће се наследити од основног ша лона, као што је приказано у [Примеру 3-9](#).

Пример 3-9. темплатес/усер.хтмл: поједностављени ша лон странице који користи наслеђивање ша лона

```
{% проширује " base.html %}

{% лоцк титле %}Фласки{% енд лоцк %}

{% лоцк паге_контент %}
<див класс="паге-хеадер">
    <x1>Здраво, {{ наме }}!</x1>
<див> {% енд лоцк %}
```



Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит џхеџкуот Зц да исте проверили ову верзију апликације.

## Линкови

Свака апликација која има више од једне руте ће увек морати да укључи везе које повезују различите странице, као што је трака за навигацију.

Писање УРЛ адреса као веза директно у ша лону је тривијално за једноставне руте, али за динамичке руте са променљивим деловима може ити компликованије за прављење УРЛ адреса

право у ша лону. Такође, УРЛ-ови написани експлицитно стварају нежељену зависност од рута дефинисаних у коду. Ако се руте реорганизују, везе у ша лонима могу покварати.

Да и се из егли ови про леми, Flask о ез еђује помоћну функцију `url_for()`, која генерише УРЛ адресе из информација усклаиштених у УРЛ мапи апликације.

У својој најједноставнијој употреби, ова функција узима назив функције приказа (или име крајње тачке за руте дефинисане са `app.add_url_rule()`) као једини аргумент и враћа њен УРЛ. На пример, у тренутној верзији хелло.пи позив `url_for('index')` и вратио /, основни УРЛ апликације. Позивање `url_for('index', _external=True)` и уместо тога вратило апсолутни УРЛ, који је у овом примеру `http://localhost:5000/`.



Релативне УРЛ адресе су довољне када се генеришу везе које повезују различите руте апликације. Апсолутне УРЛ адресе су неопходне само за везе које ће се користити изван ве претраживача, као што је када се везе шаљу путем е-поште.

Динамички УРЛ-ови се могу генерисати помоћу `url_for()` преношењем динамичких делова као аргумента кључне речи. `name='john', url_for('user')` и вратио `http://localhost:5000/`

Аргументи кључних речи послати у `url_for()` нису ограничени на аргументе које користе динамичке руте. Функција ће додати све аргументе који нису динамички стрингу упита. На пример, `url_for('user', name='john', page=2, version=1)` и вратио `/user/john?page=2&version=1`.

## Статиц Филес

Ве апликације нису направљене само од Питхон кода и ша лона. Већина апликација такође користи статичне датотеке као што су слике, ЈаваСкрипт изворне датотеке и ЦСС датотеке на које се све упућује из ХТМЛ кода у ша лонима.

Можда се сећате да када је УРЛ мапа апликације хелло.пи прегледана у [Поглављу 2](#), у њој се појавио статички унос. Flask аутоматски подржава статичке датотеке додавањем посе не руте у апликацију дефинисану као `/статиц<филенаме>`. На пример, позив `url_for('статиц', филенаме='цсс/стилес.цсс', _external=True)` и вратио `http://localhost:5000/статиц/цсс/стилес.цсс`.

У својој подразумеваној конфигурацији, Flask тражи статичке датотеке у поддиректоријуму који се зове `статиц` који се налази у основној фасциклу апликације. Датотеке се могу организовати у поддиректоријумима унутар ове фасцикли ако желите. Када сервер прими УРЛ који се пресликова на статичку руту, он

генерише одговор који укључује садржај одговарајуће датотеке у систему датотека.

**Пример 3-10** показује како апликација може да укључи икону фавицон.ико у основни шаблон да и прегледачи приказали у траци за адресу.

Пример 3-10. темплетес/ace.html: фавицон де нитион

```
{% глава лока %}
{{ супер() }} <LINK
рел="схортцут икон" хреф="{{ урл_фор('статиц', 'филенаме='фавицон.ико') }}" типе="имаге/к-икон"> <ЛИНК рел="икон" хреф="{{ урл_фор('статиц', 'филенаме='фавицон.ико') }}"
типе="имаге/к-икон">
{% енд лоцк %}
```

Декларација иконе је уметнута на крају лока главе. Одржите пажњу на то како се супер() користи за очување оригиналног садржаја лока дефинисаног у основним шаблонима.



Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит цхецкоут Зд да исте проверили ову верзију апликације.

## Локализација датума и времена са фласк-моментом

Руковање датумима и временом у веб-апликацији није тривијалан проблем када корисници раде у различитим деловима света.

Серверу су потреће не униформне временске јединице које су независне од локације сваког корисника, тако да се онично користи координирано универзално време (УТЦ). За кориснике, међутим, гледање времена израженог у УТЦ-у може ити због уњујуће, јер корисници увек очекују да виде датуме и времена представљене у њиховом локалном времену и форматирани у складу са очијима њиховог региона.

Елегантно решење које омогућава серверу да ради искључиво у УТЦ-у је да пошаље ове временске јединице у веб-претраживач, где се конвертују у локално време и приказују помоћу JavaСкрипта. Веб-прегледачи могу да ураде много овог посао у овом задатку јер имају приступ временским зонама и подешавањима локализације на рачунару корисника.

Постоји одлична библиотека отвореног кода написана на JavaСкрипту која приказује датуме и времена у претраживачу под називом **Момент.js**. Фласк-Момент је проширење за Фласк апликације које олакшава интеграцију Момент.js у Јава2 шаблоне. Фласк Момент је инсталлиран са пип-ом:

```
(веб) $ пип инсталл фласк-момент
```

Екstenзија се иницијализује на сличан начин као и Флакс-Боотстррап. Потребан код је приказан у [примеру 3-11](#).

Пример 3-11. хелло.пи: иницијализација Флакс-Момента

```
фром flask_момент импорт Момент
момент = Момент(апп)
```

Флакс-Момент зависи од `jКуери.јс` поред `Момент.јс`. Ове две и лиотеке треба да буду укључене негде у ХТМЛ документу – или директно, у ком случају можете да изaberete које верзије желите да користите, или преко помоћних функција које пружа екстензија, које упућују на тестиране верзије ових и лиотека са мреже за испоруку садржаја (`ЦДН`). Пошто Боотстррап већ укључује `јКуери.јс`, у овом случају треба додати само `Момент.јс`. [Пример 3-12](#) показује како се ова и лиотека учитава у лок скрипти ша лона, уз истовремено очување оригиналног садржаја лока који остављају основни ша лон. Имајте на уму да пошто је ово унапред дефинисан лок у основном ша лону Флакс-Боотстррап, локација у темплатес/ `ace.хтмл` где је овај лок уметнут није итна.

Пример 3-12. темплатес/ `ace.хтмл`: увоз и лиотеке `Момент.јс`

```
{% лок скрипти %}
{{ супер() }}
{{ момент.инклуде_момент() }} {%  
енд лоцк %}
```

Да и радио са временским ознакама, Флакс-Момент чини ојекат тренутка доступним ша лонима. [Пример 3-13](#) показује прослеђивање променљиве зване `цуррент_тиме` у ша лон за рендеровање.

Пример 3-13. хелло.пи: додавање променљиве датума и времена

```
фром датетиме импорт датетиме
```

```
@апп.роуте('/')
деф индек():
    ретурн рендер_темплате('индек.хтмл',
        цуррент_тиме=датетиме.утцнов())
```

[Пример 3-14](#) показује како се приказује ова променљива ша лона `цуррент_тиме`.

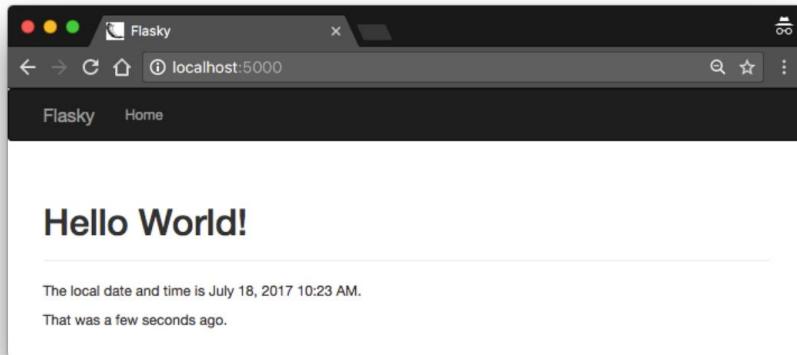
Пример 3-14. темплатес/индек.хтмл: приказивање временске ознаке помоћу Флакс-Момента

```
<п>Локални датум и време су {{ момент(цуррент_тиме).формат('ЛЛЛ') }}.</п> <п>То је ило {{ момент(цуррент_тиме).формат(рефресх=True) }}</п>
```

Функција формат('ЛЛЛ') приказује датум и време према подешавањима временске зоне и локализације на клијентском рачунару. Аргумент одређује стил приказивања, од 'Л' до 'ЛЛЛЛ' за четири различита нивоа опширности. Функција формат() такође може прихватити дугачку листу прилагођених спецификација формата.

ФромНов () стил приказивања приказан у другом реду приказује релативну временску ознаку и аутоматски је освежава како време пролази. У почетку ће ова временска ознака или приказана као „пре неколико секунди“, или опција рефресх=Труе ће је одржавати ажурираном како време пролази, тако да ако оставите страницу отворену неколико минута, видећете да се текст мења „пре минут“, затим „пре 2 минута“ и тако даље.

**Слика 3-3** показује како изгледа ruta `http://localhost:5000/` након што се две временске ознаке додају у шапон индек.хтмл.



Слика 3-3. Страница са две временске ознаке Флакс-Момент



Ако сте клонирали Гит спремиште апликације на ГитХубу, можете покренути гит цхецкоут Зе да проверите ову верзију апликације.

Флакс-Момент имплементира методе формат(), фромНов(), фромТиме(), цалендар(), валуеОф() и уник() из Момент.јс. Погледајте [документацију Момент.јс](#) да сазнате о свим опцијама форматирања које нуди овај и лиотека.



Флакс-Момент претпоставља да су временске ознаке којима рукује апликација на страни сервера „наивни“ о јекти датума и времена изражени у УТЦ-у. Погледајте документацију за [пакет датума и времена](#) у стандардној и лиотеки за информације о наивним и свесним о јектима датума и времена.

Временске ознаке које приказује Флакс-Момент могу се локализовати на многе језике. Језик се може иза рати у ша лону преношењем [двословног кода језика](#) да функционише лоцале(), одмах након укључивања и лиотеке Момент.js. На пример, ево како да конфигуришете Момент.js да користи шпански:

```
{% лок скрипти %}  
{{ супер() }}  
{{ момент.инклуде_момент() }}  
{{ момент.лоцале('es') }} {%  
енд лоц %}
```

Уз све технике о којима се говори у овом поглављу, тре ало и да удете у могућности да направите модерне ве странице прилагођене кориснику за своју апликацију. Следеће поглавље се дотиче аспекта ша лона о којима се још није расправљало: како комуницирати са корисником путем ве о разазаца.

## ГЛАВА 4

### Be Formс

Ша лони са којима сте радили у [Поглављу 3](#) су једносмерни, у смислу да омогућавају проток информација од сервера до корисника. За већину апликација, међутим, постоји и потреба за информацијама које теку у другом правцу, при чему корисник даје податке које сервер прихвата и орађује.

Са ХТМЛ-ом, могуће је креирати [вe форме](#), у које корисници могу да уносе информације. Податке о расца затим ве претраживач шаље серверу, оично у облику ПОСТ захтева. Овакат захтева Фласк, уведен у [Поглављу 2](#), излаже све информације које клијент шаље у захтеву и, посебно за ПОСТ захтеве који садрже податке о расца, омогућава приступ корисничким информацијама преко реквест форм.

Иако је подршка ове технологије у Фласковом овековеченом захтеву довољна за руковање већих расцима, постоји низ задатака који могу постати заморни и понављајући.

Два до три примера су генерирање ХТМЛ кода за формулатаре и валидација достављених података о расца.

Фласк [-BTФ](#) проширење чини рад са већим расцима много пријатнијим искуством. Ово проширење је омотач за интеграцију Фласк-а око оквира агностичких [BTForms](#) пакет.

Фласк-BTФ и његове зависности могу се инсталирати помоћу пип-а:

```
(венв) $ pip install flask-btf
```

## Цон гуратион

За разлику од већине других екстензија, Флакс-ВТФ не мора да се иницијализује на нивоу апликације, али очекује да апликација има конфигурисан тајни кључ. Тајни кључ је низ са ило којим случајним и јединственим садржајем који се користи као кључ за шифровање или потписивање да и се по ољшала ез једност апликације на неколико начина. Флакс користи овај кључ да заштити садржај корисничке сесије од неовлашћеног приступа.

Тре ало и да иза ерете другачији тајни кључ у свакој апликацији коју направите и да се уверите да овај низ није познат никоме. [Пример 4-1](#) показује како да конфигуришете тајни кључ у Флакс апликацији.

[Пример 4-1. хелло.пи: Флакс-ВТФ конфигурација](#)

```
апп = Флакс(__name__)
апп.ционфиг['СЕЦРЕТ_КЕИ'] = 'тешко погодити стринг'
```

Речник апп.ционфиг је место опште намене за складиштење конфигурационих променљивих које користи Флакс, екстензије или сама апликација. Вредности конфигурације се могу додати о јекту апп.ционфиг коришћењем стандардне синтаксе речника. Конфигурациони о јекат такође има методе за увоз вредности конфигурације из датотека или окружења. Практичнији начин управљања конфигурационим вредностима за већу апликацију иће разматран у [7. поглављу](#).

Флакс-ВТФ захтева да се тајни кључ конфигурише у апликацији јер је овај кључ део механизма који екстензија користи за заштиту свих о разаца од напада фалсификовања захтева на више локација (ЦСРФ). ЦСРФ напад се дешава када злонамерна ве локација шаље захтеве серверу апликација на који је корисник тренутно пријављен. Флакс-ВТФ генерише ез једносне токене за све о расце и чува их у корисничкој сесији, која је заштићена криптографским потписом генерисаним из тајни кључ.



За додатну сигурност, тајни кључ тре а да уде ускладиштен у променљивој окружењу уместо да уде уграђен у изворни код. Ова техника је описана у [поглављу 7](#).

## Форм Џлассес

Када користите Флакс-ВТФ, сваки ве о разац је представљен на серверу класом која наслеђује класу ФлаксФорм. Класа дефинише листу поља у о расцу, од којих је свако представљено о јектом. Сваки о јекат поља може имати један или више приложених валидатора. Валидатор је функција која проверава да ли су подаци које је корисник доставио исправни.

**Пример 4-2** приказује једноставан вео разац који има поље за текст и дугме за слање.

Пример 4-2. хелло.пи: дефиниција класе форме

```
фром фласк.втф импорт ФласкФорм фром
втформс импорт СтингФиелд, Су митФиелд фром
втформс.валидаторс импорт ДатаРекуиред

класс НамеФорм(ФласкФорм):
    наме = СтингФиелд('Како се зовеш?', валидаторс=[ДатаРекуиред()])
    су мит = Су митФиелд('Пошаљи')

    Помоћу ове функције можемо да користимо валидаторе у вредностима полја.
```

Поља у овом расцу су дефинисана као променљиве класе, а свакој променљивој класе се додељује овај експресија која је повезана са типом поља. У овом примеру, овај разац НамеФорм има текстуално поље које се зове име и дугме за слање које се зове су мит. Класа СтингФиелд представља ХТМЛ <инпут> елемент са атрибутом типе="текст". Класа Су митФиелд представља ХТМЛ <инпут> елемент са атрибутом типе="су мит". Први аргумент конструкторима поља је ознака која ће се користити приликом приказивања овог расца у ХТМЛ-у.

Опциони аргумент валидатора укључен у конструктор СтингФиелд дефинише листу провера који ће бити примењен на податке које је корисник доставио пре него што буду прихваћени. ДатаРекуиред() валидатор осигурује да поље није послато празно.



Основна класа ФласкФорм је дефинисана екstenзијом Фласк-ВТФ, тако да је увезена из фласк\_втф. Поља и валидатори се, међутим, увозе директно из пакета ВТФормс.

Листа стандардних ХТМЛ поља које подржава ВТФормс приказана је у табели 4-1.

Табела 4-1. ВТФормс стандардна ХТМЛ поља

Врста поља	Опис
БоолеанФиелд	Поље за потврду са вредностима Тачно и Нетачно
ДатеФиелд	Поље за текст које прихвата вредност датетиме.дате у датом формату
ДатетимеФиелд	Поље за текст које прихвата вредност датетиме.датетиме у датом формату
ДецималФиелд	Текстуално поље које прихвата децималну вредност. Поље за отпремање датотеке
ФилеФиелд	Скривено поље за текст
ХидденФиелд	МултиплеФилеФиелд Поље за отпремање више датотека
Листа поље	Списак поља датог типа

Врста поља	Опис
ФлоатФиелд	Текстуално поље које прихвата вредност са помичним зарезом
ФормФиелд	О разац је уградњен као поље у о разац контејнера
ИнтегерФиелд	Текстуално поље које прихвата цело ројну вредност
ПассвордФиелд	Поље за текст лозинке
РадиоФиелд	Листа радио дугмади
СелеџФиелд	Падајућа листа из ора
СелеџМултиплеФиелд	Падајућа листа из ора са вишеструким из ором
Су митФиелд	Дугмe за подношење о расца
СтрингФиелд	Текстуално поље
ТектАреаФиелд	Вишелинијско текстуално поље

Листа ВТФормс уградњених валидатора је приказана у **табела 4-2.**

#### Табела 4-2. ВТФормс валидатори

Валидатор	Опис
ДатаРекуиред	Потврђује да ли поље садржи податке након конверзије типа
Емаил	Потврђује адресу е-поште
Једнако	Упоређује вредности два поља; корисно када се захтева да се лозинка унесе два пута за потврда
ИнпутРекуиред	Потврђује да ли поље садржи податке пре конверзије типа
ИП адреса	Потврђује ИПв4 мрежну адресу
Дужина	Потврђује дужину унетог низа
МАЦ адреса	Потврђује МАЦ адресу
Нум ерРанге	Потврђује да је унета вредност унутар нумеричког опсега
Опционо	Омогућава празан унос у пољу, прескачући додатне валидаторе
Регекп	Проверава унос регуларног израза
УРЛ	Потврђује УРЛ
УУИД	Потврђује УУИД
Било који од	Потврђује да је унос једна на листи могућих вредности
Ниједна од	Потврђује да се унос не налази на листи могућих вредности

## ХТМЛ приказивање о разаца

Поља о расца су позиви који се, када се позову из ша лона, приказују у ХТМЛ-у. Под претпоставком да функција прегледа проследи инстанцу НамеФорм ша лону као аргумент са именом форм, ша лон може да генерише једноставан ХТМЛ о разац на следећи начин:

```
<форм метод="ПОСТ">
{{ форм.хидден_таг() }}
{{ форм.наме.ла ел }} {{ форм.наме() }}
{{ форм.су мит() }} </форм>
```

Имајте на уму да поред поља за име и подношење, о разац има елемент форм.хидден\_таг(). Овај елемент дефинише додатно поље о расца које је скривено, а користи га Флакс-ВТФ за имплементацију ЦСРФ заштите.

Наравно, резултат приказивања ве о расца на овај начин је крајње огњен. Сви аргументи кључне речи додати позивима који приказују поља се конвертују у ХТМЛ атријуте за поље—тако да, на пример, можете дати ид поља или атријуте класе и затим дефинисати ЦСС стилове за њих:

```
<форм метод="ПОСТ">
{{ форм.хидден_таг() }}
{{ форм.наме.ла ел }} {{ форм.наме(ид='ми-текст-фиелд') }}
{{ форм.су мит() }} </форм>
```

Али чак и са ХТМЛ атријутима, напор потрећан да и се о разац приказао на овај начин и да и изгледао до ро је значајан, тако да је најоле искористити Бootстррап-ов сопствени скуп стилова о расца кад год је то могуће. Екstenзија Флакс-Боотстррап овезује помоћну функцију високог нивоа која приказује цео Флакс-ВТФ о разац користећи Боотстррап унапред дефинисане стилове о расца, све са једним позивом. Користећи Флакс-Боотстррап, претходни о разац се може приказати на следећи начин:

```
{% импорт "оотстррап/втф.хтмл" као втф %}
{{ втф.куицк_форм(форм) }}
```

Директива увоза функционише на исти начин као и оличне Питхон скрипте и омогућава да се елементи ша лона увезу и користе у многим ша лонима. Увезена датотека оотстррап/втф.хтмл дефинише помоћне функције које приказују Флакс-ВТФ форме користећи Боотстррап. Функција втф.куицк\_форм() узима ојекат о расца Флакс-ВТФ и приказује га користећи подразумеване Боотстррап стилове. Комплетан ша лон за хелло.пи је приказан у [примеру 4-3](#).

Пример 4-3. темплатес/индек.хтмл: коришћење Флакс-ВТФ и Флакс-Боотстррап за приказивање о расца

```
{% проширује " асе.хтмл"
%} {% импорт " оотстррап/втф.хтмл" као втф %}

{% лоцк титле %}Фласки{% енд лоцк %}

{% лоцк паге_цонтент %}
<див класс="паге-хеадер">
<x1>Здраво, {% иф наме }{{ наме }}{% елсе %}Странац{% ендиф %}!</x1> </див>
{{ втф.куицк_форм(форм) }} {% енд лоцк %}
```

О ласт садржаја ша лона сада има два одељка. Први одељак је заглавље странице које приказује поздрав. Овде се користи условни ша лон. Услови у Јиња2 имају формат {% иф услов %}...{% елсе %}...{% ендиф %}. Ако је услов процењен на Тачно, онда се оно што се појављује између иф и елсе директиве додаје у приказани ша лон. Ако је услов процењен на Фалсе, онда се уместо тога приказује оно што је између елсе и ендиф . Сврха овога је да прикаже Здраво, {{ наме }}! када је дефинисана променљива ша лона имена или стринг Здраво, странче! када није. Други одељак садржаја приказује форму НамеФорм користећи функцију втф.куицк\_форм() .

## Руковање о расцима у функцијама приказа

У новој верзији хелло.пи, функција приказа индек () ће имати два задатка. Прво ће приказати о разац, а затим ће примити податке о расца које је унео корисник.

**Пример 4-4** приказује ажурирану функцију приказа индек() .

Пример 4-4. хелло.пи: руковати ве о расцем помоћу ГЕТ и ПОСТ метода захтева

```
@апп.роуте('/', методс=['ГЕТ', 'ПОСТ']) деф
индек():
    име = Нема
    форм = НамеФорм()
    ако форм.валидате_он_су мит():
        име = форм.наме.дата
        форм.наме.дата =
    ретурн рендер_темплате('индек.хтмл', форм=форм, наме=име)
```

Аргумент метод додат декоратору апп.роуте говори Флаксу да региструје функцију приказа као руковаоца за ГЕТ и ПОСТ захтеве у УРЛ мапи. Када методе нису дате, функција прегледа је регистрована за руковање само ГЕТ захтевима.

Добавање ПОСТ -а на листу метода је неопходно јер се подношењем о расца много лакше рукује као ПОСТ захтевима. Могуће је поднети о разац као ГЕТ захтев, али пошто ГЕТ захтеви немају тело, подаци се додају УРЛ адреси као стринг упита и постају видљиви у адресној траци претраживача. Из овог и неколико других разлога, подношење о расца се скоро универзално оавља као ПОСТ захтеви.

Локална променљива имена се користи за чување имена примљеног из о расца када је доступно; када име није познато, променљива се иницијализује на Ништа. Функција прегледа креира инстанцу класе НамеФорм која је претходно приказана да представља форму. Метод валидате\_он\_су мит() о расца враћа Труе када је о разац достављен и када су сви валидатори поља прихватили податке. У свим осталим случајевима, валидате\_он\_су мит() враћа Фалс. Повратна вредност овог метода ефективно служи да одреди да ли о разац трећа да се прикаже или о ради.

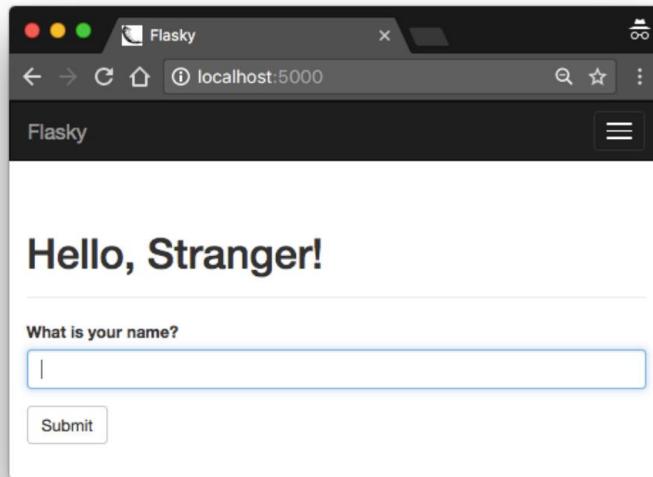
Када корисник први пут дође до апликације, сервер ће примити ГЕТ захтев са података о расца, тако да ће валидате\_он\_су мит() вратити Фалс. Тело иф наред је ће ити прескочено и захтев ће ити о рађен приказивањем ша лона, који до ија о јекат форме и променљиву наме постављене на Ништа као аргументе. Корисници ће сада видети о разац приказан у претраживачу.

Када корисник пошаље о разац, сервер до ија ПОСТ захтев са подацима. Позив валидате\_он\_су мит() позива валидатор ДатаРекуиред() који је приододат пољу имена. Ако име није празно, валидатор га прихвата и валидате\_он\_су мит() враћа Тачно. Сада је име које је унео корисник доступно као атријут података поља. Унутар тела иф наред је, ово име се додељује променљивој локалног имена и поље о расца се реше постављањем тог атријута података на празан стринг, тако да је поље празно када се о разац поново прикаже на страници. Позив рендер\_темплате() у последњем реду приказује ша лон, али овог пута аргумент наме садржи име из о расца, тако да ће поздрав ити персонализован.

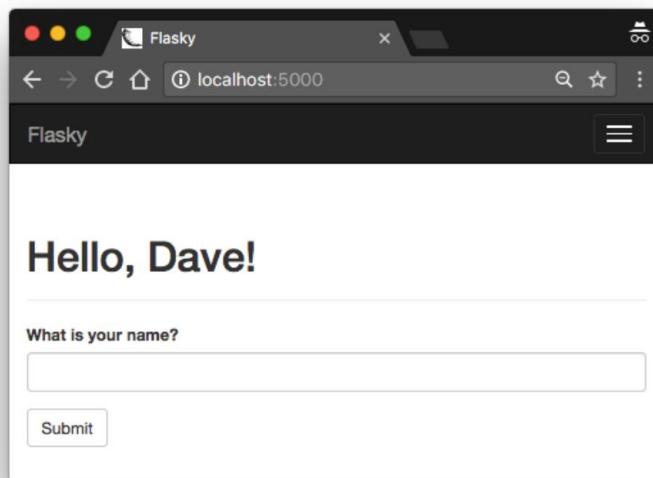


Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит цхецкоут 4а да исте проверили ову верзију апликације.

**Слика 4-1** показује како о разац изгледа у прозору претраживача када корисник уђе на сајт. Када корисник унесе име, апликација одговара персонализованим поздравом. О разац се и даље појављује испод њега, тако да корисник може да га пошаље више пута са различитим именима ако жели. **Слика 4-2** приказује апликацију у овом стању.

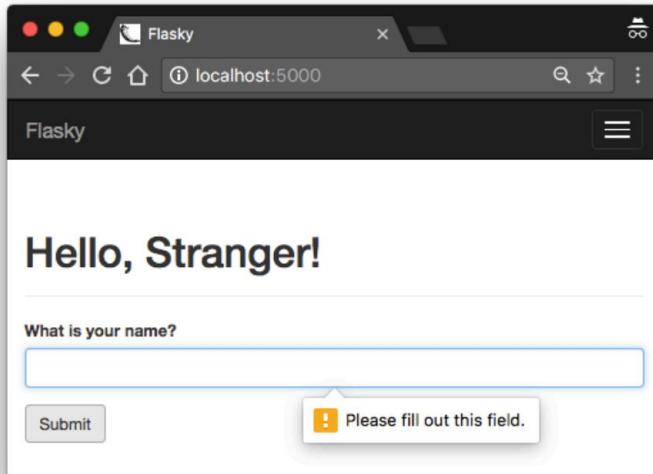


Слика 4-1. Флакс-ВТФ ве о разац



Слика 4-2. Ве о разац за подношење

Ако корисник пошаље о разац са празним именом, `dataRequired()` валидатор хвата грешку, као што се види на [слици 4-3](#). Оратите пажњу на то колико функција се аутоматски пружа. Ово је одличан пример мочи коју доноси дизајнирана проширења попут Флакс-ВТФ и Флакс-Боотстреп могу дати вашој апликацији.



Слика 4-3. Види о разац је неуспели валидатор

### Преусмеравања и корисничке сесије

Последња верзија хелло.пи има проблем са употребом ливошћу. Ако унесете своје име и пошаљете га, а затим кликнете на дугме за освежавање у претраживачу, вероватно ћете доћи или нејасно упозорење које тражи потврду пре него што поново пошаљете о разац. Ово се дешава зато што прегледачи понављају последњи захтев који су послали када се од њих тражи да освеже страницу. Када је последњи послати захтев ПОСТ захтев са подацима о расца, освежавање је изазвало подношење дупликата о расца, што у скоро свим случајевима није жељена радња. Из тог разлога, претраживач тражи потврду од корисника.

Многи корисници не разумеју ово упозорење претраживача. Сходно томе, сматра се дојном праксом да већина апликације никада не остављају ПОСТ захтев као последњи захтев који прегледач шаље.

Ово се постиже тако што се на ПОСТ захтеве одговара преусмеравањем уместо нормалног одговора. Преусмеравање је посебан тип одговора који садржи УРЛ уместо стринга са ХТМЛ кодом. Када прегледач прими одговор за преусмеравање, издаје а

ГЕТ захтев за УРЛ за преусмеравање, а то је страница коју приказује. Страница може потрајати још неколико милисекунди да се учита због другог захтева који се мора послати серверу, али осим тога, корисник неће видети никакву разлику. Сада је последњи захтев ГЕТ, тако да команда за освежавање ради како се очекује. Овај трик је познат као о разац Пост/Редиреџт/Гет.

Али овај приступ доноси други проблем. Када апликација ради ПОСТ захтев, има приступ имену које је корисник унео у форм.наме.дата, али чим се тај захтев заврши подаци о расца се губе. Пошто се ПОСТ захтевом рукује преусмеравањем, апликација треба да сачува име како би преусмерени захтев могао да га има и да га користи за креирање стварног одговора.

Апликације могу да „памте“ ствари из једног захтева у други тако што их чувају у корисничкој сесији, приватном складишту које је доступно сваком повезаном клијенту. Корисничка сесија је представљена у **поглављу 2** као једна од варијабли повезаних са контекстом захтева. Зове се сесија и приступа се као стандардном Питхон речнику.



Подразумевано, корисничке сесије се чувају у колачићима на страни клијента који су криптографски потписани помоћу конфигурисаног тајног кључа. Свако мењање садржаја колачића учинило је и потпис неважећим, чиме је и сесија поништила.

**Пример 4-5** приказује нову верзију функције приказа индекс() која имплементира преусмеравања и корисничке сесије.

Пример 4-5. хелло.пи: преусмеравања и корисничке сесије

из `флак` импорт `Флакс`, `рендер_темплате`, `сессион`, `редиреџт`, `урл_фор`

```
@апп.роуте('/', методс=['ГЕТ', 'ПОСТ']) деф
индек(): форм = НамеФорм() иф
    форм.валидате_он_суумит(): сессион['наме']
    = форм.наме.дата ретурн
        редиреџт(урл_фор('индек')) ретурн
        рендер_темплате('индек.хтмл',
            форм=форм, наме=сессион.гет('наме'))
```

У претходној верзији апликације, локална променљива имена је коришћена за чување имена које је корисник унео у о разац. Та променљива је сада смештена у корисничку сесију као сессион['наме'] тако да се памти након захтева.

Захтеви који долазе са важећим подацима о расца сада ће се завршавати позивом редиреџт(), помоћне функције Флакс која генерише ХТТП одговор преусмеравања. Функција редиреџт() узима УРЛ адресу на коју треба да преусмери као аргумент. УРЛ за преусмеравање који се користи у овом случају је основни УРЛ, тако да је одговор могао бити написан сажетије као редиреџт('/'), али се уместо тога користи Флакс-ова функција за генерисање УРЛ-а урл\_фор(), представљена у [поглављу 3](#).

Први и једини потребни аргумент за урл\_фор() је име крајње тачке, интерно име које свака ruta има. Подразумевано, крајња тачка руте је име функције приказа која је повезана са њом. У овом примеру, функција приказа која рукује основним УРЛ-ом је индек(), тако да је име дато урл\_фор() индек.

Последња промена је у функцији рендер\_темплате(), која сада добија аргумент наме директно из сесије користећи сесион.гет('наме'). Као и код редовних речника, коришћење гет() за захтевање кључа речника из егава изузетак за кључеве који нису пронађени. Метод гет() враћа подразумевану вредност Ноне за кључ који недостаје.



Ако сте клонирали Гит спремиште апликације на ГитХубу, можете покренути гит џхецкоут 4 да исте проверили ову верзију апликације.

Са овом верзијом апликације, можете видети да освежавање странице у вашем претраживачу увек резултира очекиваним понашањем.

## Порука трепери

Понекад је корисно дати кориснику ажурирање статуса након што је захтев завршен. Ово може бити порука потврде, упозорење или грешка. Типичан пример је када пошаљете о разац за пријаву на већу локацију са грешком, а сервер одговори тако што поново прикаже о разац за пријаву са поруком изнад њега која вас обавештава да су ваше корисничко име или лозинка неважећи.

Флакс укључује ову функционалност као основну функцију. [Пример 4-6](#) показује како се функција фласх() може користити за ову сврху.

Пример 4-6. хелло.пи: фласхед поруке

```
из flask импорт Flask, render_template, session, redirect, url_for, flask
```

```
@app.route('/', methods=['GET', 'POST']) деф
индек(): форм = НамеФорм() иф
форм.валидате_он_су мит():

    олд_наме = сессион.гет('наме') ако
    олд_наме није None и олд_наме != форм.наме.дата: flask('Изгледа
        да сте променили име!') сессион['наме'] = форм.наме.дата:
    ретурн redirect(url_for('индек')) ретурн
    render_template('индек.хтмл',

форм = форм, наме = сессион.гет('наме'))
```

У овом примеру, сваки пут када је име послато, оно се упоређује са именом сачуваним у корисничкој сесији, које ће тамо бити стављено током претходног подношења истог објекта. Ако су два имена различита, функција flask() се позива са поруком која се приказује на следећем одговору који се шаље назад клијенту.

Позивање flask() није доволично да би се поруке приказале; шаблон које апликација користи треба да прикаже ове поруке. Најбоље место за приказивање флаш порука је основни шаблон, јер ће то омогућити ове поруке на свим страницама. Flask чини функцију get\_flashed\_messages() доступном шаблонима за преузимање порука и њихово приказивање, као што је приказано у Примеру 4-7.

Пример 4-7. темплатес/ base.html: приказивање флашованих порука

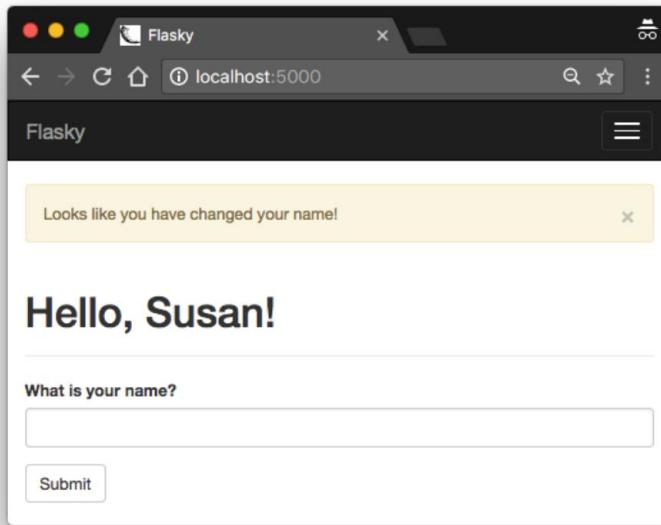
```
{% лоцк цонтент %}
<див класс="цонтainer">
    {% за поруку у get_flashed_messages() %} <див
    класс="алерт алерт-варнинг">
        < уттон типе="уттон" класс="цлосе" дата-дисмисс="алерт">&тимес;</уттон>
    {{ порука }} </див> {% ендфор %}

```

```
{% лоцк паге_цонтент %}{% енд лоцк %}
</див> {% енд лоцк %}
```

У овом примеру, поруке се приказују коришћењем Bootstrapping-ових CSS стилова упозорења за поруке упозорења (једна је приказана на слици 4-4).



Слика 4-4. Блеска порука

Петља се користи јер може ити више порука у реду за приказ, по једна за сваки пут када је `flash()` позван у претходном циклусу захтева. Поруке које су преузете из `get_flashed_messages()` неће ити враћене следећи пут када се ова функција позове, тако да се флеш поруке појављују само једном и затим се од ацују.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 4ц да исте проверили ову верзију апликације.

Могућност прихватања података од корисника путем ве о разаца је функција коју захтева већина апликација, као и могућност складиштења тих података у трајном складишту. Коришћење аза података са Фласком је тема следећег поглавља.

## ГЛАВА 5

### Базе података

База података складиши податке апликације на организован начин. Апликација затим поставља упите за преузимање одређених делова података по потребе и. Најчешће коришћене азе података за ве апликације су оне засноване на релационом моделу, које се називају и СКЛ азе података у односу на језик структурираних упита који користе.

Али последњих година, азе података оријентисане на документе и азе података кључ-вредност, неформално познате заједно као НоСКЛ азе података, постале су популарне алтернативе.

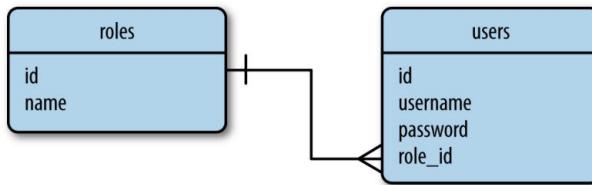
#### СКЛ азе података

Релационе азе података чувају податке у та елама, које моделирају различите ентитете у домену апликације. На пример, аза података за апликацију за управљање наруџинама ће вероватно имати та еле купаца, производа и поруџина.

Та ела има фиксни рој колона и променљив рој редова. Колоне дефинишу атријуте података ентитета представљеног та елом. На пример, та ела купаца ће имати колоне као што су име, адреса, телефон и тако даље. Сваки ред у та ели дефинише стварни елемент података који додељује вредности неким или свим колонама.

Та еле имају посебну колону која се зове примарни кључ, која садржи јединствени идентификатор за сваки ред усклађен у та ели. Та еле такође могу имати колоне које се називају страни кључеви, а који упућују на примарни кључ реда у истој или другој та ели. Ове везе између редова се називају односи и представљају основу модела релационе азе података.

**Слика 5-1** приказује дијаграм једноставне азе података са две та еле које чувају кориснике и корисничке улоге. Линија која повезује две та еле представља однос између та ела.



Слика 5-1. Пример релационе азе података

Овај графички стил представљања структуре азе података назива се дијаграм односа ентитета. У овој репрезентацији, кутије представљају та еле азе података, приказујући листе атриута или колона та еле. Та ела улога чува листу свих могућих корисничких улога, од којих је свака идентификована јединственом вредношћу ИД -а – примарним кључем та еле. Та ела корисника садржи листу корисника, сваки са својим јединственим ИД -ом . Поред примарних кључева ид , та ела улога има колону имена , а та ела корисника има колоне корисничког имена и лозинке .

Колона роле\_ид у та ели корисника је страни кључ. Линија која повезује колоне ролес.ид и усерс.роле\_ид представља однос између две та еле. Симболи причвршћени за линију на сваком крају указују на кардиналност односа. На страни ролес.ид , линија је приказана као „један”, док је на страни усерс.роле\_ид представљено „много”. Ово приказује однос један према више, што указује да сваки ред из та еле улога може ити повезан са много редова из

та ела корисника .

Како што се види у примеру, релационе азе података ефикасно складиште податке и избегавају дуплирање. Преименовање корисничке улоге у овој ази података је једноставно јер имена улога постоје на једном месту. Одмах након што се име улоге промени у та ели улога , сви корисници који имају роле\_ид који упућује на промењену улогу видеће ажурирање.

С друге стране, поделити податке у више та ела може ити компликација.

Израда листе корисника са њиховим улогама представља мали проблем, јер корисници и корисничке улоге морају ити прочитани из две та еле и спојени пре него што се могу представити заједно. Машина релационих аза података пружају подршку за оављање операција спајања између та ела када је то потребе но.

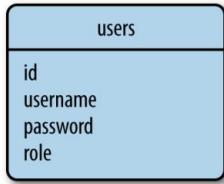
## НоСКЛ азе података

Базе података које не прате релациони модел описан у претходном одељку заједнички се називају НоСКЛ азама података. Једна уочијена организација за НоСКЛ азе података користи колекције уместо та ела и документе уместо записа.

НоСКЛ азе података су дизајниране на начин да отежава спајање, тако да већина њих уопште не подржава ову операцију. За НоСКЛ азу података структурирану као на [слици 5-1](#),

листање корисника са њиховим улогама захтева да сама апликација изврши операцију придрживања читањем поља `role_id` сваког корисника, а затим претражујући тај елу улога за то.

Прикладнији дизајн за НоСКЛ је података је приказан на [слици 5-2](#). Ово је резултат примене операције која се зове денормализација, која смањује број табела на рачун дуплицирања података.



Слика 5-2. Пример НоСКЛ табеле података

База података са овом структуром има име улоге експлицитно сачувано код сваког корисника.

Преименовање улоге се тада може показати као скупа операција која може захтевати ажурирање великог броја докумената.

Али нису све лоше вести са НоСКЛ табелама података. Дуплирање података омогућава једностављавање упита. Навођење корисника и њихових улога је једноставно јер нису потребне табеле придрживања.

## СКЛ или НоСКЛ?

СКЛ табеле података су одличне у складиштењу структурираних података у ефикасном и компактном облику. Ове табеле података се труде да очувају конзистентност, чак и у случају нестанка струје или хардверских квркова. Парадигма која омогућава релационим табелама података да достигну овај висок ниво поузданости назива се [АЦИД](#), што је скраћеница за атомичност, конзистентност, изолацију и трајност. НоСКЛ табеле података опуштају неке од АЦИД захтева и као резултат тога понекад могу да имају предност у перформансама.

Потпуна анализа и поређење типова табела података је ван оквира ове књиге. За мале и средње апликације, СКЛ и НоСКЛ табеле података су савршено способне и имају практично једнаке перформансе.

## Питхон оквири табеле података

Питхон има пакете за већину мотора табела података, како отвореног кода, тако и комерцијалних. Флакс не поставља ограничења на то који пакети табеле података могу да се користе, тако да можете да радите са МиСКЛ, Постгрес, СКЛите, Редис, МонгоДБ, ЦоуцхДБ или ДинамоДБ ако вам је потребно који од ових омиљених.

Како да то није довољан из ор, постоји и велики рој пакета слојева апстракције азе података, као што су СКЛАлцхеми или МонгоЕнгине, који вам омогућавају да радите на вишем нивоу са регуларним Питхон ојектима уместо ентитета азе података као што су та еле, документи или језици упита.

Постоји неколико фактора које треја проценити када ирате оквир азе података:

#### Лакоћа

употреје Када се пореде директне машине азе података са слојевима апстракције азе података, друга група очигледно по еђује. Слојеви апстракције, који се такође називају ојектно-релациони мапери (ОРМ) или ојектно-документни мапери (ОДМ), ојези еђују транспарентну конверзију ојеката оријентисаних операција високог нивоа у инструкције азе података ниског нивоа.

#### Перформанс

Конверзије које ОРМ-ови и ОДМ-ови морају да ураде да и превели са домена ојекта у домен азе података имају прекомерне трошкове. У већини случајева, казна за учинак је занемарљива, или можда није увек. Генерално, повећање продуктивности до ијено са ОРМ-овима и ОДМ-овима далеко надмашије минималну деградацију перформанси, тако да ово није валидан аргумент да се ОРМ-ови и ОДМ-ови потпуно од аце.

Оно што има смисла је одајати слој апстракције азе података који ојези еђује опциони приступ основној ази података у случају да специфичне операције треба да се оптимизују директном имплементацијом као инструкција изворне азе података.

#### Преносивост

Морају се узети у обзир изори азе података доступни на вашим развојним и производним платформама. На пример, ако планирате да хостујете своју апликацију на платформи у објаку, требају ало и да сазнате које изоре азе података нуди ова услуга.

Други аспект преносивости се односи на ОРМ и ОДМ-ове. Иако неки од ових оквира ојези еђују слој апстракције за један механизам азе података, други апстрахују још више и пружају изор механизама азе података—свима њима се може приступити са истим ојектно оријентисаним интерфејсом. Најбољи пример за то је СКЛ-Алцхеми ОРМ, који подржава листу машина за релационе азе података укључујући популарни МиСКЛ, Постгрес и СКЛите.

#### Интеграција са

Фласком Одајир оквира који има интеграцију са Фласком није апсолутно ојавезан, али ће вас уштедети од потреје да сами пишете интеграциони код. Интеграција фласк-а и могла да поједностави конфигурацију и рад, тако да и требају дати предност коришћењу пакета посебно дизајнираног као проширење фласке.

На основу ових циљева, иза рани оквир азе података за примере у овој књизи иће [Фласк-СКЛАлцхеми](#), омотач проширења Фласк за СКЛАлцхеми.

## Управљање азом података помоћу Фласк-СКЛАЛЦХЕМИ

Фласк-СКЛАЛЦХЕМИ је проширење за Фласк које поједностављује употребу СКЛАЛЦХЕМИ унутар Фласк апликација. СКЛАЛЦХЕМИ је моћан оквир релационе азе података који подржава неколико позадинских делова азе података. Нуђи ОРМ високог нивоа и приступ на ниском нивоу изворној СКЛ функционалности азе података.

Како и већина других екстензија, Фласк-СКЛАЛЦХЕМИ се инсталира са пип:

(венв) \$ пип инсталл фласк-склалцхеми

У Фласк-СКЛАЛЦХЕМИ, аза података је наведена као УРЛ. Та ела 5-1 наводи формат УРЛ адреса за три најпопуларнија механизма аза података.

Та ела 5-1. УРЛ-ови азе података Фласк-СКЛАЛЦХЕМИ

Дата асе енгине	УРЛ
Мискл	мискл://усернаме:пассворд@хостнаме/дата асе
Постгрес	постгрескл://усернаме:пассворд@хостнаме/дата асе
СКЛите (Линук, маџОС)	склите:///а_солуте/патх/то/дата асе
СКЛите (Виндовс)	склите:///ц:а_солуте/патх/то/дата асе

У овим УРЛ адресама, име хоста се односи на сервер који хостује услугу азе података, што може ити локални или удалjeni сервер. Сервери аза података могу да хостују неколико аза података, тако да аза података указује на име азе података коју тре а користити. За азе података којима је потре на аутентификација, корисничко име и лозинка су кориснички акредитиви азе података.



СКЛите азе података немају сервер, тако да су име хоста, корисничко име и лозинка изостављени, а аза података је име датотеке на диску за азу података.

УРЛ азе података апликације мора ити конфигурисан као кључ СКЛАЛЦХЕМИ\_ДАТАБАСЕ\_УРИ у конфигурационом ојекту Фласк. Документација Фласк-СКЛАЛЦХЕМИ такође предлаже постављање кључа СКЛАЛЦХЕМИ\_ТРАЦК\_МОДИФИЦАЦИОНС на Фалсе да и се користило мање меморије осим ако нису потре ни сигнали за промене ојектата. Консултујте Фласк СКЛАЛЦХЕМИ документацију за информације о другим опцијама конфигурације.

Пример 5-1 показује како да иницијализујете и конфигуришете једноставну СКЛите азу података.

Пример 5-1. хелло.пи: конфигурација азе података

```
импорт ос
фром фласк_склалцхеми импорт СКЛАЛЦХЕМИ
```

```
аседир = ос.патх.а спатх(ос.патх.дирнаме(__филе__))
```

```

апп = Flask(__name__)
апп.конфиг['СКЛАЛЦХЕМИ_ДАТАБАСЕ_УРИ'] = 'склите://'
    + ос.патх.join( аседир, 'дата.склите')
апп.конфиг['СКЛАЛЦХЕМИ_ТРАЦК_МОДИФИЦАЦИОНС'] = Нетачно

```

д = СКЛАЛцхеми(апп)

О јекат д инстанциран из класе СКЛАЛцхеми представља азу података и о ез еђује приступ свим функционалностима Флакс-СКЛАЛцхеми.

## Модел Де нитион

Термин модел се користи када се односи на трајне ентитетете које апликација користи. У контексту ОРМ-а, модел је типично Питхон класа са атри утима који одговарају колонама одговарајуће та еле азе података.

Инстанца азе података из Флакс-СКЛАЛцхеми о ез еђује основну класу за моделе као и скуп помоћних класа и функција које се користе за дефинисање њихове структуре. Та еле улога и корисника са [слике 5-1](#) могу се дефинисати као модели Улога и Корисник као што је приказано у Примеру 5-2.

Пример 5-2. хелло.пи: Дефиниција модела улоге и корисника

`класс Роле(д .Модел):`

```

_та _ленаме_ = 'ролес' ид =
д .Цолумн(д .Интигер, примари_кей =Труе) наме =
д .Цолумн(д .Стринг(64), јединствено=Труе)

```

```

деф __repr__(селф): ретурн
    '<Улога %p>' % селф.наме

```

`класс Усер(д .Модел):`

```

_та _ленаме_ = 'усерс' ид =
д .Цолумн(д .Интигер, примари_кей =Труе) корисничко име
= д .Цолумн(д .Стринг(64), јединствено=Труе, индек =Труе)

```

```

деф __repr__(селф):
    ретурн '<Корисник %p>' % селф.усернаме

```

Променљива класе \_та \_ленаме\_ дефинише име та еле у ази података. Флакс СКЛАЛцхеми додељује подразумевано име та еле ако је \_та \_ленаме\_ изостављено, али та подразумевана имена не прате популарну конвенцију коришћења множине за имена та ела, тако да је нај оље да се та еле именују експлицитно. Преостале варија ле класе су атри ути модела, дефинисани као инстанце класе д .Цолумн .

Први аргумент дат конструктору д .Цолумн је тип колоне азе података. умн и атри ут модела. Та ела 5-2 наводи неке од типова колона који су доступни, заједно са типовима Питхон-а који се користе у моделу.

### Та ела 5-2. Најчешћи типови колона СКЛАЛцхеми

Унесите име	Питхон тип	Опис
Интегер	инт	Регуларни цео рој, о ично 32 ита
СмаллИнтегер инт		Цео рој кратког домета, о ично 16 ита
БигИнтегер	инт или лонг	Неограничен цео рој прецизности
Пловак	пловак	Број са помичним зарезом
Нумериц	децимални.Децимални	Број фиксне тачке
Низ	стр	Стринг променљиве дужине
Текст	стр	Стринг променљиве дужине, оптимизован за велику или неограничену дужину
Уницоде	уникод	Уницоде стринг променљиве дужине
УницодеТект уникод		Уницоде стринг променљиве дужине, оптимизован за велику или неограничену дужину
Боолеан	оол	Боолеан валуе
Датум	датетиме.дате	Вредност датума
време	датетиме.тиме	Временска вредност
Датум време	датетиме.датетиме	Вредност датума и времена
Интервал	датетиме.тимеделта	Временски интервал
Енум	стр	Листа вредности стрингова
Пицклетипе	Било који Питхон о јекат	Аутоматска серијализација Пицкле
Ларгебинари стр		Бинарна мрља

Преостали аргументи д .Цолумн специфицирају опције конфигурације за сваки атри ут. Та ела 5-3 наводи неке од доступних опција.

### Та ела 5-3. Најчешће опције СКЛАЛцхеми колоне

Име опције	Опис
примарни_кључ	Ако је постављено на Тачно, колона је примарни кључ та еле.
јединствени	Ако је подешено на Тачно, не дозволите дупле вредности за ову колону.
индекс	Ако је постављено на Тачно, креирајте индекс за ову колону, тако да су упити ефикаснији.
нулла_ле	Ако је постављено на Тачно, дозволите празне вредности за ову колону. Ако је постављено на Фалсе, колона неће дозволити нулте вредности.
уо_ичајено	Дефинишите подразумевану вредност за колону.



Флак-СКЛАлцхеми захтева да сви модели дефинишу колону примарног кључа, која се оично назива ид.

Иако то није стриктно неопходно, ова два модела укључују метод `_репр_()` који им даје читљиву репрезентацију стрингова која се може користити у сврхе отклањања грешака и тестирања.

## ОДНОСИ

Релационе азе података успостављају везе између редова у различитим таелама коришћењем релација. Релациони дијаграм на [слици 5-1](#) изражава једноставан однос између корисника и њихових улога. Ово је однос један-према-више од улога до корисника, јер једна улога може припадати многим корисницима, али сваки корисник може имати само једну улогу.

**Пример 5-3** показује како је однос један-према-више на [слици 5-1](#) представљен у класама модела.

Пример 5-3. хелло.пи: односи у моделима азе података

```
класс Роле(д .Модел):
    # Корисници =
    д .релационсхип('Усер', ацкреф='роле')
```

```
класс Усер(д .Модел):
    # Роле_ид =
    д .Цолумн(д .Интегер, д .ФореигнКеи('ролес.ид'))
```

Како што се види на [слици 5-1](#), однос повезује два реда коришћењем страног кључа. Колона `роле_ид` дodata моделу корисника дефинисана је као страни кључ и то успоставља однос. Аргумент 'ролес.ид' за `д .ФореигнКеи()` специфицира да колону трећа тумачити као да има вредности ИД-а из редова у тајели улога.

Атриут корисника који је додат моделу Улога представља објектно оријентисан поглед на однос, гледано са стране „један“. С овиром на инстанцу класе Роле, атриут усерс ће вратити листу корисника повезаних са том улогом (тј. страна „много“). Први аргумент за `д .релационсхип()` указује који је модел на другој страни односа. Класа модела може ити дата као стринг ако је класа дефинисана касније у модулу.

Аргумент `ацкреф` за `д .релационсхип()` дефинише ограничени правац односа, додајућем атриута улоге моделу корисника. Овај атриут се може користити

на ило којој инстанци корисника уместо страног кључа роле\_ид да исте приступили моделу улоге као о јекту.

У већини случајева д .релатионсхип() може сам да лоцира страни кључ везе, али понекад не може да одреди коју колону да користи као страни кључ. На пример, ако модел корисника има две или више колона дефинисаних као страни кључеви улоге , онда СКЛАЛцхеми не и знао коју од две колоне да користи. Кад год је конфигурација страног кључа двосмислена, тре а дати додатне аргументе д .релатионсхип() . Та ела 5-4 наводи неке од уо ичјених опција конфигурације које се могу користити за дефинисање односа.

#### Та ела 5-4. Уо ичјене опције СКЛАЛцхеми односа

Назив опције	Опис Додајте
ацкреф	повратну референцу у други модел у односу.
примариоин	Експлицитно наведите услов спајања између два модела. Ово је неопходно само за двосмислене односе.
лењ	Одредите како ће се повезане ставке учитати. Могуће вредности су иза ране (ставке се учитавају на захтев када им се први пут приступи), непосредне (ставке се учитавају када се изворни о јекат учита), спојене (ставке се учитавају одмах, али као спој), потупит (ставке се учитавају одмах , али као потупит), нолоад (ставке се никада не учитавају) и динамички (уместо учитавања ставки, даје се упит који их може учитати).
уселист	Ако је постављено на Фалсе, користите скалар уместо листе.
ордер_ и	Наведите редослед који се користи за ставке у односу.
секундарни	Одредите име та еле асоцијације коју ћете користити у релацијама много-према-више.
секондариоин	Одредите секундарни услов спајања за релације много-према-много када СКЛАЛцхеми не може сам да га одреди.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 5а да исте проверили ову верзију апликације.

Постоје и други типови односа осим један-према-више. Однос један-на-један може се изразити на исти начин као један-према-више, као што је раније описано, али са опцијом листе корисника постављеном на Фалсе унутар дефиниције д .релатионсхип () тако да страна „много“ постаје „једне стране. Однос више-према-један се такође може изразити као један-према-више ако су та еле о рнуте, или се може изразити са страним кључем и дефиницијом д .релатионсхип() на страни „много“. Најсложенији тип односа, много-према-више, захтева додатну та елу која се зове та ела асоцијације или спојна та ела. У 12. поглављу ћете научити о односима много-према-многима .

## Операције азе података

Модели су сада у потпуности конфигурисани према дијаграму азе података на [слици 5-1](#) и спремни су за употребу. Најоли начин да научите како да радите са овим моделима је у Питхон љусци. Следећи одељци ће вас провести кроз најчешће операције азе података у љусци започетој командом фласк скел . Пре него што употребите ову команду, уверите се да је променљива окружења ФЛАСК\_АПП подешена на хелло.пи, као што је приказано у [поглављу 2](#).

### Креирање та ела Прва

ствар коју треба да урадите је да упутите Фласк-СКЛАлцхеми да креира азу података на основу класа модела. Функција д.цреате\_алл() лоцира све подкласе д.Модел и креира одговарајуће та еле у ази података за њих:

```
(венв) $ фласк скелл
>>> из хелло импорт д
>>> д.цреате_алл()
```

Ако проверите директоријум апликације, видећете нову датотеку која се зове `дата.склите`, име које је дато СКЛите ази података у конфигурацији. Функција д.цреате\_алл() неће поново креирати или ажурирати та елу азе података ако она већ постоји у ази података. Ово може ити незгодно када се модели модификују и промене треба применити на постојећу азу података. Решење групе сила за ажурирање постојећих та ела азе података на другу шему је да прво уклоните старе та еле:

```
>>> д.дроп_алл()
>>> д.цреате_алл()
```

Нажалост, овај метод има нежељени нежељени ефекат уништавања свих података у старој ази података. Боље решење про лема ажурирања аза података представљено је при kraju поглавља.

### Уметање редова

Следећи пример креира неколико улога и корисника:

```
>>> из хелло импорт Улога, корисник
>>> админ_роле = Улога(наме='Админ')
>>> мод_роле = Улога(наме='Модератор')
>>> усер_роле = Улога(наме='Корисник') >>
> усер_жохн = Корисник(усернаме='жохн', улога=админ_роле)
>>> усер_сусан = Усер(усернаме='сусан', роле=усер_роле) >>>
усер_давид = Усер(усернаме='давид', роле=усер_роле)
```

Конструктори за моделе прихватају почетне вредности за атрибуте модела као аргументе кључне речи. Имајте на уму да се атрибут роле може користити, иако то није права колона азе података, већ представљање високог нивоа односа један-према-више. ид -

атри ут ових нових ојеката није експлицитно постављен: примарним кључевима у многим азама података управља сама аза података. Ојекти за сада постоје само на Питхон страни; још нису уписаны у азу података. Зато гој тога њихове ид вредности још увек нису додељене:

```
>>> print(admin_role.id)
```

Ниједан

```
>>> print(mod_role.id)
```

Ништа >>> print(user\_role.id)

Ниједан

Променама азе података се управља кроз сесију азе података, коју Флакс СКЛАЛцхеми ојез еђује као д.сесијон. Да исте припремили ојекте за писање у азу података, они морају ити додати сесији:

```
>>> d.session.add(admin_role)
>>> d.session.add(mod_role) >>>
d.session.add(user_role) >>>
d.session.add(user_john) >>>
d.session.add(user_susan) >>>
d.session.add(user_david)
```

Или, сажетије:

```
>>> d.session.add_all([admin_role, mod_role, user_role,
...                     user_john, user_susan, user_david])
```

Да исте уписали ојекте у азу података, сесија мора ити урезана позивањем њеног цоммит() метода:

```
>>> d.session.commit()
```

Поново проверите ид атриуте након што сте предали податке да исте видели да ли јесу сада постављено:

```
>>> print(admin_role.id)
1 >>> print(mod_role.id)
2
```

```
>>> print(user_role.id)
3
```



Сесија азе података д.сесијон није повезана са ојектом сесије Флакс о којем се говори у [поглављу 4](#). Сесије азе података се такође називају трансакцијама.

Сесије азе података су изузетно корисне у одржавању конзистентности азе података. Операција урезивања записује атомски све ојекте који су додати сесији. Ако

грешка се јавља док се сесија уписује, цела сесија се од ацује. Ако увек урезујете повезане промене заједно у сесији, гарантовано ћете из ећи недоследности азе података з ог делимичних ажурирања.



Сесија азе података се такође може вратити.

Ако се позове д .сесион.ролл ацк() , сви о јекти који су додати сесији азе података се враћају у стање које имају у ази података.

## Модификовање

редова Адд() метод сесије азе података се такође може користити за ажурирање модела. Настављајући у истој сесији љуске, следећи пример преименује улогу „Админ“ у „Администратор“:

```
>>> админ_роле.наме = 'Администратор' >>>
д .сесион.адд(админ_роле) >>>
д .сесион.цоммит()
```

## Брисање редова

Сесија азе података такође има методу делете() . Следећи пример прише улогу „Модератор“ из азе података:

```
>>> д .сесион.делете(мод_роле) >>>
д .сесион.цоммит()
```

Имајте на уму да се брисања, попут уметања и ажурирања, извршавају само када је сесија азе података урезана.

## Куериинг Ровс

Флак-СКЛАЦХеми чини о јекат упита доступан у свакој класи модела. Најосновнији упит за модел се покреће методом алл() , која враћа цео садржај одговарајуће та еле:

```
>>> Роле.куери.алл()
[<Улога 'Администратор', <Улога 'Корисник'>]
>>> Усер.куери.алл()
[<Корисник 'joхн', <Корисник 'сусан', <Корисник 'давид'>]
```

О јекат упита се може конфигурисати да издаје специфичније претраге азе података коришћењем лтера. Следећи пример проналази све кориснике којима је додељена улога „Корисник“ :

```
>>> Усер.куери.фильтер_ и(роле=усер_роле).алл()
[<Корисник 'сусан', <Корисник 'давид'>]
```

Такође је могуће прегледати изворни СКЛ упит који СКЛАЛцхеми генерише за дати упит претварањем о јекта упита у стринг:

```
>>> стр(Усер.куери.филтер_ и(роле=усер_роле))
'СЕЛЕЦТ усерс.ид АС усер_ид, усерс.усернаме АС усер_усернаме,
усерс.роле_ид АС усерс_роле_ид \нФРОМ корисника \нВХЕРЕ :парам_1 = усерс.роле_ид'
```

Ако изађете из сесије љуске, о јекти креирани у претходном примеру ће престати да постоје као Питхон о јекти, али ће наставити да постоје као редови у одговарајућим табелама азле података. Ако затим започнете потпуно нову сесију љуске, морате поново да креирате Питхон о јекте из њихових редова азле података. Следећи пример издаје упит који учитава корисничку улогу са именом „Корисник“:

```
>>> усер_роле = Роле.куери.филтер_ и(наме='Усер').фирст()
```

Одјратите пажњу на то како је у овом случају упит издат са методом фирмст() уместо алл(). Док алл() враћа све резултате упита као листу, фирмст() враћа само први резултат или Ништа ако нема резултата, тако да је згодан метод за употребу уз упите за које се зна да враћају један резултат на већину.

Филтери као што је филтер\_ и() се позивају на о јекту упита и враћају нови прецишћени упит. Више филтера се може позвати у низу док се упит не конфигурише по потреби и.

**Тајка 5-5** приказује неке од најчешћих филтера доступних за упите. Комплетна листа је у СКЛАЛцхеми документацији.

Тајка 5-5. Уочијени СКЛАЛцхеми упити лтерс

Опција	Опис
филтер()	Враћа нови упит који додаје додатни филтер оригиналном упиту
филтер_ и()	Враћа нови упит који додаје додатни филтер једнакости оригиналном упиту
лимит()	Враћа нови упит који ограничава број резултата оригиналног упита на дати број
офсет()	Враћа нови упит који примењује помак на листу резултата оригиналног упита
ордер_ и()	Враћа нови упит који сортира резултате оригиналног упита према датим критеријумима
груп_ и()	Враћа нови упит који групише резултате оригиналног упита према датим критеријумима

Након што су жељени филтери примењени на упит, позив алл() ће изазвати извршење упита и вратити резултате као листу—али постоје и други начини да се покрене извршење упита осим алл(). **Тајка 5-6** приказује друге методе извршења упита.

## Тајела 5-6. Најчешћи СКЛАДЦИ извршиоци упита

Опција	Опис
све()	Враћа све резултате упита као листу
први()	Враћа први резултат упита или Нема ако нема резултата
фирст_оп_404()	Враћа први резултат упита или прекида захтев и шаље грешку 404 као одговор ако постоји нема резултата
до_ити()	Враћа ред који одговара датом примарном кључу или Ништа ако није пронађен ниједан одговарајући ред
гет_оп_404()	Враћа ред који одговара датом примарном кључу или, ако кључ није пронађен, поништава захтев и шаље грешку 404 као одговор
коунт()	Враћа број резултата упита
пагинирати()	Враћа објекат Пагинације који садржи наведени опсег резултата

Релације функционишу слично као и упити. Следећи пример испитује однос један према више између улога и корисника са ове краја:

```
>>> корисници = усер_роле.усерс
>>> корисници
[<Корисник 'сусан'>, <Корисник 'давид'>]
>>> корисници[0].улога
<Улога 'Корисник'>
```

Упит усер\_роле.усерс овде има проблем. Имплицитни упит који се покреће када се израз усер\_роле.усерс изда интерно позива алл() да врати списак корисника. Пошто је објекат упита скривен, са њим није могуће прецизирати додатни филтери упита. У овом конкретном примеру, можда је ишло корисно затражити да се листа корисника врати по агенцијском реду. У примеру 5-4, конфигурација односа је модификован аргументом лењи='динамички' да ће и се захтевало да се упит се не извршава аутоматски.

Пример 5-4. хелло.пи: динамички односи азе података

Улога `класе` (десни .Модел):

```
...
# корисника = десни .релационсхип('Усер', ацкеф='роле', лази='динамиц')
# ...
```

Са односом конфигурисаним на овај начин, усер\_роле.усерс враћа упит који се још увек није извршио, тако да му се могу додати филтери:

```
>>> усер_роле.усерс.ордер_и(Усер.усернаме).алл()
[<Корисник 'давид'>, <Корисник 'сусан'>]
>>> усер_роле.усерс.коунт()
2
```

## Коришћење азе података у функцијама приказа

Операције азе података описане у претходним одељцима могу се користити директно унутар функција приказа.  
**Пример 5-5** приказује нову верзију руте почетне странице која елеки имена која су корисници унели у азу података.

Пример 5-5. хелло.пи: употреба азе података у функцијама приказа

```
@апп.роуте('/' метходс=['ГЕТ', 'ПОСТ']) деф
индек(): форм = НамеФорм() иф
форм.валидате_он_су мит():

    усер = Усер.куери.фильтер_ и(усернаме=форм.наме.дата).фирст() ако је
    корисник Ништа: усер = Усер(усернаме=форм.наме.дата)
        д .сесион.адд(усер) д .сесион.коммит () сесион['кновн'] =
        Нетачно другачије: сесион['кновн'] = Тачна сесија['наме'] =
        форм.наме.дата форм.наме.дата =
```

```
""
    врати преусмеравање(урл_фор('индек'))
ретурн рендер_темплате('индек.хтмл',
    форм=форм, наме=сесион.гет('наме'),
    кновн=сесион.гет('кновн', Фалсе))
```

У овој измененој верзији апликације, сваки пут када се име пошаље, апликација га проверава у ази података помоћу филтера упита филтер\_ и(). Позната променљива се уписује у корисничку сесију тако да се након преусмеравања информације могу послати у ша лон, где се користе за прилагођавање поздрава. Имајте на уму да да и апликација радила, та еле азе података морају ити креирање у Питхон љусци као што је раније приказано.

Нова верзија придруженог ша лона је приказана у **Примеру 5-6**. Овај ша лон користи познати аргумент за додавање другог реда у поздрав који се разликује за познате и нове кориснике.

Пример 5-6. темплатес/индек.хтмл: прилагођени поздрав у ша лону

```
{% проширује " асе.хтмл" %}
{% импорт " оотстрап/втф.хтмл" као втф %}

{% лоцк титле %}Фласки{% енд лоцк %}

{% лоцк паге_цонтент %}
<див цлас="паге-хеадер">
<x1>Здраво, {% иф наме %}{{ наме }}{% еlse %}Странац{% ендиф %}</x1> {% иф нот
кновн %}
```

```
<п>Драго ми је што сам вас упознао!
</п> {% елсе %} <п>Драго ми је да вас
поново видим!</п> {%- ендиф %} </див>
{{ втф.куицк_форм(форм) }} { % крајни
лок %}
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 5 да исте проверили ову верзију апликације.

## Интеграција са Питхон Схелл-ом

Увоз инстанце азе података и модела сваки пут када се покрене сесија љуске је напоран посао. Да исте из егли стално понављање ових корака, команда фласк схелл се може конфигурисати да аутоматски увози ове ојекте. Да исте додали ојекте на листу увоза, процесор контекста љуске мора ити креиран и регистрован у декоратору апп.схемл\_цонтект\_процессор . Ово је приказано у [Примеру 5-7.](#)

Пример 5-7. хелло.пи: додавање контекста љуске

```
@апп.схемл_цонтект_процессор деф
make_схемл_цонтект(): ретурн
    дикт(д = д , Усер=Усер, Роле=Роле)
```

Функција процесора контекста враћа речник који укључује инстанцу азе података и моделе. Команда фласк схелл ће аутоматски увести ове ставке у љуску, поред апликације, која се подразумевано увози:

```
$ фласк схелл
>>> апп
<Фласк 'здраво'>>>
д
<СКЛАлчеми енгине='склите:///хоме/фласк/фласки/дата.склите'>>> Корисник
<класа 'хелло.Усер'>
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 5ц да исте проверили ову верзију апликације.

## Миграције азе података са Фласк-Миграте

Како напредујете у развоју апликације, открићете да модели азе података морају да се промене, а када се то деси и аза података тре а да се ажурира. Фласк-СКЛАлцхеми креира та еле азе података из модела само када они већ не постоје, тако да је једини начин да се ажурирају та еле тако да се прво униште старе та еле—али наравно, то доводи до губитка свих података у ази података.

Боље решење је коришћење оквира за миграцију азе података. На исти начин на који алати за контролу верзија извornог кода прате промене у датотекама извornог кода, оквир за миграцију азе података прати промене у шеми азе података, омогућавајући примену инкременталних промена.

Програмер СКЛАлцхеми је написао оквир за миграцију под називом [Алем иц](#), али уместо да директно користе Алем иц, Фласк апликације могу користити [Фласк-Миграте](#) проширење, лагани Алем иц омот који га интегрише са командом фласк .

### Креирање миграционог спремишта

За почетак, Фласк-Миграте мора ити инсталацији у виртуелном окружењу:

(венв) \$ pip инсталал фласк-миграте

[Пример 5-8](#) показује како се екстензија иницијализује.

Пример 5-8. хелло.пи: Иницијализација Фласк-Миграте

фром [фласк\\_миграте импорт Миграте](#)

```
# ...
```

```
миграција = Миграција(апликација, д...)
```

Да и изложио команде миграције азе података, Фласк-Миграте додаје фласк д команду са неколико подкоманди. Када радите на новом пројекту, можете додати подршку за миграције азе података помоћу подкоманде инит :

(венв) \$ фласк д инит

Креирање директоријума /хоме/фласк/фласки/мигратионс...готово  
 Креирање директоријума /хоме/фласк/фласки/мигратионс/версионс...готово  
 Генерирање /хоме/фласк/фласки/мигратионс/алем иц.ини ...готово  
 Генерирање /хоме/фласк/фласки/мигратионс/енв.пи...дона Генерирање /хоме/фласк/фласки/мигратионс/енв.пи...дона Генерирање /хоме/фласк/фласки/мигратионс/РЕАДМЕ...готово Генерирање /хоме/фласк/фласки/мигратионс/скрипт.пи.мако...дона Уредите подешавања конфигурације/везе/регистровања у '/хоме/фласк/фласки/мигратионс/алем иц.ини' пре него што наставите.

Ова команда креира директоријум за миграције, где ће ити ускладиштене све скрипте за миграцију. Ако пратите пример пројекта користећи гит цхецкоут, не морате да радите овај корак, пошто је спремиште за миграцију већ укључено у ГитХу спремиште.



Датотеке у спремишту за миграцију азе података увек морају да се додају контроли верзија заједно са остатком апликације.

### Креирање скрипте за миграцију У

Алем ицу, миграција азе података је представљена скриптом за миграцију. Ова скрипта има две функције које се зову упграде() и довнграде(). Функција упграде() примењује промене азе података које су део миграције, а функција довнграде() их уклања. Ова могућност додавања и уклањања промена значи да Алем иц може реконфигурисати азу података у ило којој тачки у историји промена.

Алем ичке миграције се могу креирати ручно или аутоматски користећи команде ревизије и миграције . Ручна миграција креира скрипту скелета миграције са празним функцијама упграде() и довнграде() које тре а да имплементира програмер користећи директиве изложене Алем иц-овом о јекту Оператионс .

Аутоматска миграција покушава да генерише код за функције упграде() и довнграде() тражећи разлике између дефиниција модела и тренутног стања азе података.



Аутоматске миграције нису увек тачне и могу да пропусте неке детаље који су двосмислени. На пример, ако се колона преименује, аутоматски генерисана миграција може показати да је дотична колона из рисана и да је додата нова колона са новим именом. Ако оставите миграцију каква јесте, то ће довести до гу итка података у овој колони! Из тог разлога, аутоматски генерисане скрипте за миграцију тре а увек прегледати и ручно исправити ако имају ило какве нетачности.

Да исте извршили промене у шеми азе података помоћу Флакс-Миграте-а, потре но је следити следећу процедуру:

1. Направите потре не измене у класама модела.
2. Креирајте скрипту за аутоматску миграцију помоћу команде флакс д миграте .
3. Прегледајте генерисану скрипту и прилагодите је тако да тачно представља промене које су направљене на моделима.

4. Додајте скрипту за миграцију у контролу извора.

5. Примените миграцију на азу података помоћу команде флакс д упграде .

Подкоманда флакс д миграте креира аутоматску скрипту за миграцију:

```
(веб) $ флакс д мигрирати -м "почетна миграција"
ИНФО [алем иц.мигратион] Контекст импл СКЛитеИмпл.
ИНФО [алем иц.мигратион] Претпоставиће нетрансакциони ДДЛ.
ИНФО [алем иц.аутогенерате] Откривена додата та ела 'улоге'
ИНФО [алем иц.аутогенерате] Откривена додата та ела 'корисници'
ИНФО [алем иц.аутогенерате.цомпартре] Откривен је додат индекс
'ик_усерс_усернаме' на '[усернаме]'

Генерирање /хоме/флакс/фласки/мигратионс/версионс/1 ц
594146 5_инитијал_мигратион.пи...готово
```

Ако пратите упутства за гит цхецкоут да исте постепено ажурирали пример апликације, не морате да издајете команде за миграцију , пошто су скрипте за миграцију већ уграђене у ознаке Гит спремишта.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 5д да исте проверили ову верзију апликације. Имајте на уму да не морате да генеришете миграционо спремиште и скрипте за миграцију за ову апликацију пошто су оне укључене у ГитХу спремиште.

## Надоградња азе података

Када се скрипта за миграцију прегледа и прихвати, може се применити на азу података помоћу команде за надоградњу флакс д :

```
(веб) $ флакс д упграде
ИНФО [алем иц.мигратион] Контекст импл СКЛитеИмпл.
ИНФО [алем иц.мигратион] Претпоставиће нетрансакциони ДДЛ.
ИНФО [алем иц.мигратион] Покретање надоградње Ништа -> 1 ц594146 5, почетна миграција
```

За прву миграцију, ово је ефективно еквивалентно позивању д .цреате\_алл(), али у узастопним миграцијама команда флакс д упграде применује ажурирања на та еле ез утицаја на њихов садржај.



Ако сте радили са апликацијом у њеним претходним фазама, већ имате датотеку азе података која је раније креирана помоћу функције `д_цреате_алл()`. У овом стању, надоградња фласк д неће успети јер ће покушати да креира та еле азе података које већ постоје. Једноставан начин да се реши овај проблем је да изришете датотеку азе података `дата.склите`, а затим покренете фласк д упграде да исте генерирали нову азу података кроз оквир за миграцију.

Друга опција је да прескочите надоградњу фласк д и уместо тога означите постојећу азу података као надограђену помоћу команде `фласк д стамп`.

### Добавање више миграција

Док радите на сопственим пројектима, открићете да морате врло често да правите промене у моделима азе података. Када управљате азом података преко оквира за миграцију, све промене морају ити дефинисане у скриптама за миграцију, јер све што се не прати у миграцији неће ити поновљиво. Процедура за увођење промене у азу података је слична оној која је урађена за увођење прве миграције:

1. Направите потре не промене у моделима азе података.
2. Генеришите миграцију помоћу команде `фласк д миграте`.
3. Прегледајте генерисану скрипту за миграцију и исправите је ако има ило каквих нетачности.
4. Примените промене на азу података помоћу команде `фласк д упграде`.

Док радите на одређеној функцији, можда ћете открити да морате да направите неколико промена у моделима азе података пре него што их дојнете онако како желите. Ако ваша последња миграција још увек није ила посвећена контроли извора, можете се одлучити да је проширите да исте укључили нове промене док их правите, а то ће вас уштедети од много малих скрипти за миграцију које су саме по се и есмислене. Процедура за проширење последње скрипте за миграцију је следећа:

1. Уклоните последњу миграцију из азе података помоћу команде `фласк д довнграде` (имајте на уму да то може довести до губитка неких података).
2. Изришите последњу скрипту за миграцију, која је сада без родитеља.
3. Генеришите нову миграцију азе података помоћу команде `фласк д миграте`, која ће сада укључити измене у скрипти за миграцију коју сте управо уклонили, плус све друге промене које сте направили на моделима.
4. Прегледајте и примените скрипту за миграцију као што је претходно описано.



Консултуйте [документацију](#) Флакс-Миграте да исте сазнали више о другим подкомандама у вези са миграцијама азе података.

Тема дизајна и коришћења азе података је веома важна; написане су читаве књиге на ову тему. Ово поглавље тре а да посматрате као преглед; напредније теме иће речи у каснијим поглављима. Следеће поглавље је посвећено слању е-поште.

## ГЛАВА 6

### Емайл

Многе врсте апликација морају да оавесте кориснике када се догоде одређени догађаји, а уо и чајени начин комуникације је е-маил. У овом поглављу ћете научити како да шаљете е-пошту из Флакс апликације.

#### Подршка путем е-поште са Флакс-Маил

Иако се пакет смтпли из стандардне лиотеке Питхон може користити за слање е-поште унутар Флакс апликације, проширење Флакс-Маил оавија смтпли и лепо га интегрише са Флакском. Флакс-Маил се инсталира са пип-ом:

```
(venv) $ pip install flask-mail
```

Екстензија се повезује са сервером Симпле Маил Трансфер Протоцол (СМТП) и прослеђује е-поруке на њега за испоруку. Ако није дата конфигурација, Флакс-Маил се повезује на лоцалност на порту 25 и шаље е-поштуez аутентификације. Та ела 6-1 приказује листу конфигурационих кључева који се могу користити за конфигурисање СМТП сервера.

Та ела 6-1. Кључеви за конфигурацију СМТП сервера Флакс-Маил

Кључ	Подразумевани опис	
МАИЛ_СЕРВЕР	лоцалност	Име хоста или ИП адреса сервера е-поште
МАИЛ_ПОРТ	25	Порт сервера е-поште
МАИЛ_УСЕ_ТЛС	Фалсе	Омогући ez едност транспортног слоја (ТЛС).
МАИЛ_УСЕ_ССЛ	Фалсе	Омогући ez едност слоја ez едних утичница (ССЛ).
МАИЛ_УСЕРНАМЕ	Ништа	Корисничко име налога за пошту
МАИЛ_ПАССВОРД	Ништа	Лозинка налога за пошту

Током развоја може ити згодније повезати се са екстерним СМТП сервером. Као пример, [Пример 6-1](#) показује како да конфигуришете апликацију да шаље е-пошту преко Гоогле Гмаил налога.

Пример 6-1. хелло.пи: Конфигурација Флакс-Майл-а за Гмаил

```
увоз ос #
...
апп.цонфиг['МАИЛ_СЕРВЕР'] = 'смтп.гооглемайл.цом'
апп.цонфиг['МАИЛ_ПОРТ'] = 587
апп.цонфиг['МАИЛ_УСЕ_ТЛС'] =
= Тачно
апп.цонфиг['МАИЛ_УСЕРНАМЕ'] = ос.енviron.гет
('МАИЛ_УСЕРНАМЕ') апп.цонфиг['МАИЛ_ПАССВОРД'] =
ос.енviron.гет('МАИЛ_ПАССВОРД')
```



Никада не пишите акредитиве налога директно у своје скрипте, посебно ако планирате да оставите свој рад као отворени код. Да и исте заштитили информације о налогу, нека ваша скрипта увеезе осетљиве информације из променљивих окружења.



Из езедносних разлога, Гмаил налози су конфигурисани тако да захтевају да спољне апликације користе ОАутХ2 аутентификацију за повезивање са сервером е-поште. Нажалост, Питхонова смтпли и лиотека не подржавају овај метод аутентификације. Да и ваш Гмаил налог прихватио стандардну СМТП аутентификацију, идите на страницу подешавања [Гоогле налога](#) и изаберите „Пријава на Гоогле“ на левој траци менија. На тој страници пронађите поставку „Дозволи мање езедне апликације“ и уверите се да је омогућена. Ако се омогућавање овог подешавања на вашем личном Гмаил налогу тиче вас, направите секундарни налог само да исте тестирали слање е-поште.

Флакс-Майл се иницијализује као што је приказано у [Примеру 6-2](#).

Пример 6-2. хелло.пи: Иницијализација Флакс-Майл-а

```
фром фласк_майл импорт Майл
майл = Майл(апп)
```

Две променљиве окружења које држе корисничко име и лозинку сервера е-поште морају ити дефинисане у окружењу. Ако сте на Линук-у или мацОС-у, можете подесити ове варијанте на следећи начин:

```
(венв) $ екпорт МАИЛ_УСЕРНАМЕ=<Гмаил корисничко
име> (венв) $ екпорт МАИЛ_ПАССВОРД=<Гмаил лозинка>
```

За кориснике Микрософт Виндовс-а, променљиве окружења су подешене на следећи начин:

```
(веб) $ сет МАИЛ_УСЕРНАМЕ=<Гмаил корисничко
име> (веб) $ сет МАИЛ_ПАССВОРД=<Гмаил лозинка>
```

Слање е-поште из Питхон Схелл -а Да исте тестирали

конфигурацију, можете започети сесију ЉУСКЕ и послати про ну е-пошту (замените иоу@екампле.цом својом сопственом адресом е-поште):

```
(веб) $ фласк схелл
>>> из фласк_майл импорт поруке >>>
из хелло импорт майл >>> мср =
Порука('тест емаил', сендер='иоу@екампле.цом',
...     реципиентс=['иоу@екампле.цом'])
>>> мср. оди = 'Ово је тело о иног текста' >>>
мср.хтмл = 'Ово је < >ХТМЛ</ > тело' >>> са
апп.апп_цонтект(): майл.сенд( порука)
...
...
```

Имајте на уму да функција сенд() Флакс-Майл користи цуррент\_апп, тако да тре а да се изврши са активираним контекстом апликације.

Интеграција е-порука са апликацијом Да исте из егли да

сваки пут ручно креирате поруке е-поште, до ра је идеја да у функцију апстрахујете заједничке делове функције слања е-поште апликације. Као додатна предност, ова функција може да прикаже тела е-поште са Јиња2 ша лона да имају највећи флекси илност. Имплементација је приказана у [Примеру 6-3](#).

Пример 6-3. хелло.пи: подршка путем е-поште

из [фласк\\_майл импорт поруке](#)

```
апп.ценфиг['ФЛАСКИ_МАИЛ_СУБЈЕЦТ_ПРЕФИКС'] = '[Флакси]'
апп.ценфиг['ФЛАСКИ_МАИЛ_СЕНДЕР'] = 'Флакси Админ <флакси@екампле.цом>'
```

```
деф сенд_емаил(за, предмет, ша лон, **кваргс): мср =
Порука(апп.ценфиг['ФЛАСКИ_МАИЛ_СУБЈЕЦТ_ПРЕФИКС'] + тема,
...     сендер=апп.ценфиг['ФЛАСКИ_МАИЛ_СЕНДЕР'], реципиентс=[за])
мср. оди = рендер_темплате(темплате + '.ткт', **кваргс) мср.хтмл =
...     рендер_темплате(темплате + '.хтмл', **кваргс ) майл.сенд(мср)
```

Функција се ослања на два конфигурациона кључа специфична за апликацију која дефинишу низ префикса за предмет и адресу која ће се користити као пошиљалац. Функција сенд\_емаил() узима одредишну адресу, тему, ша лон за тело е-поште и листу аргумента кључних речи. Мора се навести назив ша лона

ез екстензије, тако да се две верзије ша лона могу користити за о ичан текст и ХТМЛ тела. Аргументи кључне речи које је прослеђивао позивалац дају се рендер\_темплате() тако да их ша лони који генеришу тело е-поште могу користити као променљиве ша лона.

Функција приказа индек () може се лако проширити да пошаље е-пошту администратору кад год се уз о разац прими ново име. [Пример 6-4](#) показује ову промену.

#### Пример 6-4. хелло.пи: пример е-поште

```
...
# app.конфиг['ФЛАСКИ_АДМИН'] = os.environ.get('ФЛАСКИ_АДМИН')
# ...
@app.route('/', методы=['ГЕТ', 'ПОСТ'])
def индек( ):
    : форм = НамеФорм()
    форм.валидате_он_сү мит():

        усер = Усер.куери.фильтр_
        и(усернаме=форм.наме.дата).фирист() ако
        је корисник Ништа: усер = Усер(усернаме=форм.наме.дата)
        д .сессион.адд(усер) сессион['кновн'] = Фалсе иф
        app.конфиг['ФЛАСКИ_АДМИН']:
        сенд_емайл(app.конфиг['ФЛАСКИ_АДМИН'], 'Нови корисник',
        'маил/нев_усер', усер=усер)

    елсе:
        сессион['кновн'] = Тачна
        сесија['наме'] = форм.наме.дата
        форм.наме.дата = ретурн
        редиреџт(урл_фор('индек')) ретурн
        рендер_темплате('индек.хтмл', форм=форм, наме=сессион.гет('наме'),
        познато=сессион.гет('познато', Нетачно))
```

Прималац е-поште је дат у променљивој окружењу ФЛАСКИ\_АДМИН која се учитава у конфигурациону променљиву истог имена током покретања. Потре но је направити две датотеке ша лона за текстуалну и ХТМЛ верзију е-поште. Ове датотеке се чувају у поддиректоријуму поште унутар ша лона како и иле одвојене од о ичних ша лона. Ша лони е-поште очекују да корисник уде дат као аргумент ша лона, па га позив сенд\_емайл() укључује као аргумент кључне речи.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут ба да исте проверили ову верзију апликације.

Поред променљивих окружења МАИЛ\_УСЕРНАМЕ и МАИЛ\_ПАССВОРД описаних раније, овој верзији апликације потреће да је променљива окружења ФЛАСКИ\_АДМИН. За кориснике Линук-а и мацОС-а, команда за постављање ове променљиве је:

```
(веб) $ екпорт ФЛАСКИ_АДМИН=<ваша-е-адреса>
```

За кориснике Мицрософт Виндовс-а, ово је еквивалентна команда:

```
(веб) $ сет ФЛАСКИ_АДМИН=<ваша-е-адреса>
```

Са овим подешеним варијаблама окружења, можете тестирати апликацију и примати е-пошту сваки пут када унесете ново име у овај разац.

Слање асинхроне е-поште Ако сте

послали неколико променљивих е-порука, вероватно сте приметили да се функција mail.send() локира на неколико секунди док се е-пошта шаље, због чега прегледач не реагује за то време. Дајмо и се избегла непотребна кашњења током овог рада захтева, функција слања е-поште може да се премести у позадинску нит. **Пример 6-5** показује ову промену.

Пример 6-5. hello.py: подршка за асинхрони email

```
from txreadding import Thread
```

```
def send_asinc_email(app, msg):
```

```
    with app.app_context():
```

```
        mail.send(msg)
```

```
def send_email(za, предмет, шалон, **kwargs): msg =
```

```
    Порука(app.config['ФЛАСКИ_МАИЛ_СУБЈЕЦТ_ПРЕФИКС'] + тема,
```

```
    сендер=app.config['ФЛАСКИ_МАИЛ_СЕНДЕР'], реципиент=за])
```

```
    msg.оди = рендер_темплате(темплате + '.txt', **kwargs) msg.xtml =
```

```
    рендер_темплате(темплате + '.xhtml', **kwargs) txr = Thread(target=send_asinc_email,
```

```
    args=[app, msg]) txr.start() return txr
```

Ова имплементација наглашава занимљив проблем. Многа проширења Флакс-а раде под претпоставком да постоје активни контексти апликације и/или захтева. Као што је раније поменуто, функција send() Флакс-Майл-а користи цуррент\_апп, тако да захтева да контекст апликације буде активан. Али пошто су контексти повезани са нити, када се функција mail.send() изврши у другој нити, потребно је да се контекст апликације креира вештачки коришћењем app.app\_context(). Инстанца апликације се прослеђује нити као аргумент тако да се може креирати контекст.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут б да исте проверили ову верзију апликације.

Ако сада покренете апликацију, приметићете да много оље реагује, али имајте на уму да је за апликације које шаљу велику количину е-поште прикладније имати посао посвећен слању е-поште него покретање нове нити за сваку е-пошту послати операцију. На пример, извршење функције `сенд_асинц_емайл()` може се послати на [Целери](#) ред задатака.

Ово поглавље завршава преглед функција које су неопходне за већину ве апликација. Сада је про лем у томе што скрипта хелло.пи почиње да расте и то отежава рад са њом. У следећем поглављу ћете научити како да структурирате већу апликацију.

## ГЛАВА 7

# Велика структура апликације

Иако поседовање малих ве апликација усклаиштених у једној скрипту датотеки може ити веома згодно, овај приступ се не може до ро скалирати. Како апликација постаје све сложенија, рад са једном великом извornом датотеком постаје про лематичан.

За разлику од већине других ве оквира, Флакс не намеће посе ну организацију за велике пројекте; начин структуирања апликације је у потпуности препуштен програмеру. У овом поглављу је представљен могући начин организовања велике апликације у пакетима и модулима. Ова структура ће се користити у преосталим примерима књиге.

### Структура пројекта

**Пример 7-1** показује основни изглед за Флакс апликацију.

Пример 7-1. Основна вишеструка структура апликације Флакс

```
|-фласки
 |-апп/
   |-тэмплейтес/
   |-статиц/
   |-
   |майн/
   |-
   |__инит__.пи |
   |-
   |еррорс.пи |
   |-
   |формс.пи |
   |-
   |виеовс.пи |
   |-
   |__инит__.пи |
   |-
   |емайл.пи |
   |-
   |моделс.пи |
   |-
   |миграционс/
   |-
   |тестс/ | __инит__.пи
   |-
   |тест*.пи | венв/
   |-
   |рекуирментс.ткт
   |-
   |конфиг.пи |
   |-
   |фласки.пи
```

Ова структура има четири фасцикли највишег нивоа:

- Флакс апликација живи унутар пакета са општим именом.
- Фасцикли миграција садржи скрипте за миграцију азе података, као и раније.
- Јединични тестови су написани у пакету тестова.
- Фасцикли венв садржи Питхон виртуелно окружење, као и раније.

Постоји и неколико нових датотека:

- Захтеви.ткт наводи зависности пакета тако да је лако регенерисати идентично виртуелно окружење на другом рачунару.
- цон г.пи чува подешавања конфигурације.
- фласки.пи дефинише инстанцу апликације Флакс, а такође укључује и неколико задатака који помоћи у управљању апликацијом.

Да и вам помогли да у потпуности разумете ову структуру, следећи одељци описују процес претварања апликације хелло.пи у њу.

## Опције конфигурације

Апликацијама је често потребно неколико конфигурационих скупова. Најоли пример за то је потреба да се користе различите азе података током развоја, тестирања и производње како не и ометале једна другу.

Уместо једноставне конфигурације попут речника ап.цонфиг коју користи хелло.пи, може се користити хијерархија конфигурационих класа. [Пример 7-2](#) приказује датотеку цон г.пи, са свим подешавањима уvezеним из хелло.пи.

Пример 7-2. цон г.пи: конфигурација апликације

```
импорт ос
аседир = ос.патх.а спатх(ос.патх.дирнаме(_филе_))

цласс Цонфиг:
    СЕЦРЕТ_КЕИ = ос.енviron.гет('СЕЦРЕТ_КЕИ') или 'тешко погодити стринг'
    МАИЛ_СЕРВЕР = ос.енviron.гет('МАИЛ_СЕРВЕР', 'smtp.гооглемайл.цом')
    МАИЛ_ПОРТ = инт(ос.енviron.гет('МАИЛ_ПОРТ', '587'))
    МАИЛ_УСЕ_ТЛС = ос.енviron.гет('МАИЛ_УСЕ_ТЛС', 'true').ловер() у \
        ['тачно', 'укључено', '1']
    МАИЛ_УСЕРНАМЕ = ос.енviron.гет('МАИЛ_УСЕРНАМЕ')
    МАИЛ_ПАССВОРД = ос.енviron.гет('МАИЛ_ПАССВОРД')
    ФЛАСКИ_МАИЛ_СУБЈЕЦТ_ПРЕФИКС = '[Флакски]'
    ФЛАСКИ_МАИЛ_СЕНДЕР = 'Флакски администратор <флакски@екампле.цом>'
    ФЛАСКИ_АДМИН = ос.енviron.гет('ФЛАСКИ_АДМИН')
    СКЛАДЦХЕМИ_ТРАЦК_МОДИФИЦАЦИОНС = Нетачно
```

```
@статицметод
дев инит_апп(апп): пасс
```

класа [ДевелопментЦонфиг\(Цонфиг\)](#):

```
ДЕБУГ = Тачно
СКЛАЛЦХЕМИ_ДАТАБАСЕ_УРИ = ос.енviron.гет('ДЕВ_ДАТАБАСЕ_УРЛ') или \ 'склите:/// +
ос.патх.join( аседир, 'дата-дев.склите')
```

класа [ТестингЦонфиг\(Цонфиг\)](#):

```
ТЕСТИРАЊЕ = Тачно
СКЛАЛЦХЕМИ_ДАТАБАСЕ_УРИ = ос.енviron.гет('ТЕСТ_ДАТАБАСЕ_УРЛ') или \ 'склите://'
```

класа [ПродуционЦонфиг\(Цонфиг\)](#):

```
СКЛАЛЦХЕМИ_ДАТАБАСЕ_УРИ = ос.енviron.гет('ДАТАБАСЕ_УРЛ') или \ 'склите:/// +
ос.патх.join( аседир, 'дата.склите')
```

цонфиг =

```
{ 'девелопмент': ДевелопментЦонфиг,
'тестинг': ТестингЦонфиг, 'продуцион':
ПродуционЦонфиг,
```

'подразумевано': [ДевелопментЦонфиг](#)

}

Основна класа Цонфиг садржи подешавања која су заједничка за све конфигурације; различите подкласе дефинишу подешавања која су специфична за конфигурацију. По потребе и се могу додати додатне конфигурације.

Да и конфигурација или флекси илинија и ез једнија, већина подешавања се опционо може увести из променљивих окружења. На пример, вредност СЕЦРЕТ\_КЕИ, з ог његове осетљиве природе, може да се подеси у окружењу, али подразумевана вредност је дата у случају да је окружење не дефинише. Оично се ове поставке могу користити са својим подразумеваним вредностима током развоја, али свако тре да има одговарајућу вредност постављену у одговарајућу променљиву окружења на производном серверу. Све опције конфигурације за сервер е-поште се такође увозе из променљивих окружења, са подразумеваним вредностима које упућују на Гмаил сервер ради погодности током развоја.



Никада немојте писати лозинке или друге тајне у конфигурациону датотеку која је посвећена контроли извора.

Променљивој СКЛАЛЦХЕМИ\_ДАТАБАСЕ\_УРИ се додељују различите вредности у оквиру сваке од три конфигурације. Ово омогућава апликацији да користи различите азе података у свакој од њих

конфигурацију. Ово је веома важно, јер не желите да покретање јединичних тестова промени азу података коју користите за свакодневни развој. Свака конфигурација покушава да увезе УРЛ азе података из променљиве окружења, а када она није доступна поставља подразумевану на основу СКЛите-а. За конфигурацију тестирања, подразумевана је аза података у меморији, пошто нема потре е да се чувају подаци ван про ног покретања.

Свака конфигурација развоја и производње има скуп опција конфигурације сервера поште. Као додатни начин да се дозволи апликацији да прилагоди своју конфигурацију, класа Цонфиг и њене подкласе могу дефинисати метод класе инист\_апп() који узима инстанцу апликације као аргумент. За сада основна класа Цонфиг имплементира празну методу инист\_апп().

На дну конфигурационе скрипте, различите конфигурације су регистроване у конфигурационом речнику. Једна од конфигурација (у овом случају она за развој) је такође регистрована као подразумевана.

## Пакет апликација

Пакет апликације је место где живе сви кодови апликације, ша лони и статички фајлови. Овде се зове једноставно апликација, иако јој се по жељи може дати назив специфичан за апликацију. Ша лони и статички директоријуми су сада део пакета апликације, тако да се премештају унутар апликације. Модели азе података и функције подршке за е-пошту се такође премештају унутар овог пакета, сваки у свом модулу, као апп/моделс.пи и апп/емаил.пи.

### Коришћење фа рике апликација

Начин на који се апликација креира у верзији са једним фајлом је веома згодан, али има један велики недостатак. Пошто је апликација креирана у глобалном опсегу, не постоји начин да се промене конфигурације примене динамички: у време када се скрипта покрене, инстанца апликације је већ креирана, тако да је већ прекасно да се промене конфигурације. Ово је посебно важно за јединичне тестове јер је понекад потребно покренути апликацију под различитим поставкама конфигурације ради ове покривености тестом.

Решење овог проблема је одлагање креирања апликације премештањем у функцију која се може експлицитно позвати из скрипте. Ово не само да даје скрипти време да подеси конфигурацију, већ и могућност да креира више инстанци апликације — још једна ствар која може исти веома корисна током тестирања. Функција апликације, приказана у [примеру 7-3](#), дефинисана је у пакету апликације конструктор.

### Пример 7-3. апп/\_инит\_.пи: конструктор пакета апликације

```
фром flask импорт Flask,рендер_темплате фром
flask_оотстрап импорт Bootstrapping фром flask_mail
импорт Mail фром flask_moment импорт Moment фром
flask_skalalzhemim импорт СКЛАЛЦХЕМИ фром цонфиг
импорт цонфиг
```

```
оотстрап = Bootstrapping() mail =
Mail() момент = Moment() д =
СКЛАЛЦХЕМИ()
```

```
деф цреате_апп(цонфиг_наме): апп =
    Flask(__name__)
    апп.цонфиг.фром_о јеңт(цонфиг[цонфиг_наме])
    цонфиг[цонфиг_наме].инит_апп(апп)
```

```
    оотстрап.инит_апп(апп)
    mail.инит_апп(апп)
    момент.инит_апп(апп)
    д .инит_апп(апп)
```

# приложите руте и прилагођене странице са грешкама овде

[врати апликацију](#)

Овај конструктор увози већину Flask екстензија које се тренутно користе, али пошто не постоји инстанца апликације са којом и их иницијализовала, ствара их неиницијализоване тако што не прослеђује аргументе у њихове конструкторе. Функција цреате\_апп() је фабрика апликација, која као аргумент узима име конфигурације која се користи за апликацију. Подешавања конфигурације усклађиштена у једној од класа дефинисаних у цонг.пи могу се увести директно у апликацију коришћењем методе фром\_о јеңт() доступног у Flask-овом апп.цонфиг конфигурационом објекту. Конфигурациони објекти се ира по имени из конфигурационог речника. Када се апликација креира и конфигурише, екстензије се могу иницијализовати. Позивање инит\_апп() на екстензијама које су раније креиране довршава њихову иницијализацију.

Иницијализација апликације се сада врши у овој фабричкој функцији, користећи метод фром\_о јеңт() из конфигурационог објекта Flask, који као аргумент узима једну од конфигурационих класа дефинисаних у цонг.пи. Метода инит\_апп() иза ране конфигурације се такође позива, како и се омогућиле сложеније процедуре иницијализације.

Фабричка функција враћа креирану инстанцу апликације, али имајте на уму да су апликације креиране са фабричком функцијом у тренутном стању некомплетне, јер им недостају руте и прилагођени руководоци страницама грешака. Ово је тема следећег одељка.

## Имплементација функционалности апликације у нацрту Конверзија

у фа рику апликација уводи компликацију за руте. У апликацијама са једном скриптом, инстанца апликације постоји у гло алном опсегу, тако да се руте могу лако дефинисати помоћу декоратора апп.роуте . Али сада када је апликација креирана током извршавања, декоратор апп.роуте почиње да постоји тек након што се позове цреате\_апп() , што је прекасно. Прилагођени руковаоци страницама грешака представљају исти про лем, јер су дефинисани декоратором апп.эррорхандлер .

Срећом, Флакс нуди оље решење користећи нацрте. Нацрт је сличан апликацији по томе што такође може да дефинише руте и руковаоце грешкама. Разлика је у томе што када су они дефинисани у нацрту, они су у стању мировања док се нацрт не региструје у апликацији, у ком тренутку они постају део њега. Користећи нацрт дефинисан у гло алном опсегу, руте и руковаоци грешкама апликације могу се дефинисати на скоро исти начин као у апликацији са једном скриптом.

Као и апликације, нацрти се могу дефинисати у једној датотеци или се могу креирати на структуриранији начин са више модула унутар пакета. Да и се омогућила највећа флекси илност, иће креиран подпакет унутар пакета апликације за смештај првог плана апликације. [Пример 7-4](#) приказује конструктор пакета који креира нацрт.

Пример 7-4. апп/маин/\_инит\_.пи: креирање главног нацрта

[из флакс импорт Блуепринт](#)

```
маин = Блуепринт('маин', __наме__)
```

[од . увоз погледа, грешке](#)

Нацрти се креирају инстанцирањем објекта класе Блуепринт. Конструктор за ову класу узима два обавезна аргумента: име нацрта и модул или пакет где се нацрт налази. Као и код апликација, Питхон-ова променљива \_\_наме\_\_ је у већини случајева тачна вредност за други аргумент.

Руте апликације се чувају у модулу апп/маин/виевс.пи унутар пакета, а руковаоци грешака су у апп/маин/эррорс.пи. Увоз ових модула доводи до тога да руте и руковаоци грешака буду повезани са нацртом. Важно је напоменути да се модули увозе на дну апп/маин/\_инит\_.пи скрипте да и се избегле грешке због кружних зависности. У овом конкретном примеру проблем је у томе што ће апп/маин/виевс.пи и апп/маин/эррорс.пи заузват увести главни објекат нацрта, тако да ће увозити неуспешан осим ако се кружна референца не појави након што се маин дефинише .



Техе фром . импорт <соме-модуле> синтакса се користи у Питхон-у за представљање релативног увоза из јединице пакета. Ускоро ћете видети још један веома користан релативни увоз који користи о разац из .. импорт <соме-модула, где је родитељ тренутног пакета.

Нацрт је регистрован у апликацији унутар фа ричке функције цреате\_апп() , као што је приказано у [Примеру 7-5](#).

Пример 7-5. апп/\_инит\_.пи: регистрација главног плана

```
деф цреате_апп(конфиг_наме):
    # ...
```

```
из .маин импорт маин као маин_
    луепринт
апп.регистер_ луепринт(маин_ луепринт)
```

[врати апликацију](#)

Пример 7-6 показује руковаоце грешкама.

Пример 7-6. апп/маин/еррорс.пи: руковаоци грешака у главном нацрту

```
из флак импорт рендер_темплате из .
    увоз главни
```

```
@маин.апп_еррорхандлер(404)
деф паге_нот_фоунд(е):
    ретурн рендер_темплате('404.хтмл'), 404
```

```
@маин.апп_еррорхандлер(500)
деф интернал_сервер_еррор(е):
    ретурн рендер_темплате('500.хтмл'), 500
```

Разлика при писању руковаоца грешкама унутар нацрта је у томе што ако се користи декоратор за о раду грешака, руковалац ће ити позван само за грешке које потичу на рутама дефинисаним нацртом. Да исте инсталарирали о рађиваче грешака у целој апликацији, уместо тога се мора користити декоратор апп\_еррорхандлер .

Пример 7-7 приказује руту апликације која је ажурирана да уде у нацрту.

Пример 7-7. апп/маин/виевс.пи: руте апликације у главном нацрту

```
фром датетиме импорт датетиме
фром фласк импорт рендер_темплате, сесион, редиреџт, урл_фор
фром . импорт маин фром .формс импорт НамеФорм из .. импорт д
фром ..моделс импорт Усер
```

```
@маин.роуте('/', методс=['ГЕТ', 'ПОСТ']) деф
индек( ): форм = НамеФорм() иф
    форм.валидате_он_су мит():

# ...
    врати преусмеравање(урл_фор('.индек'))
ретурн рендер_темплате('индек.хтмл',
    форм=форм, наиме=сесион.гет('наиме'),
    кновн=сесион.гет('кновн', Фалсе),
    цуррент_тиме=датетиме.утцнов())
```

Постоје две главне разлике када се пише функција приказа унутар нацрта. Прво, као што је раније урађено за руковаоце грешкама, декоратор руте долази из нацрта, тако да се маин.роуте користи уместо апп.роуте. Друга разлика је у коришћењу функције урл\_фор(). Као што се можда сећате, први аргумент ове функције је име крајње тачке руте, које је за руте засноване на апликацији подразумевано на име функције приказа. На пример, у апликацији са једном скриптом УРЛ за функцију приказа индек( ) може се до ити помоћу урл\_фор('индек').

Разлика са нацртима је у томе што Флак примењује простор имена на све крајње тачке дефинисане у нацрту, тако да више нацрта може дефинисати функције погледа са истим именима крајњих тачака – ез колизија. Именски простор је име нацрта (први аргумент конструктора Блуупринт) и одвојен је од имена крајње тачке тачком. Функција приказа индек() се затим региструје са именом крајње тачке маин.индек и њен УРЛ се може до ити помоћу урл\_фор('маин.индек').

Функција урл\_фор() такође подржава краћи формат за крајње тачке у нацртима у којима је име нацрта изостављено, као што је урл\_фор('.индек'). Са овом нотацијом, име нацрта за тренутни захтев се користи да се заврши име крајње тачке. Ово ефективно значи да преусмеравања унутар истог плана могу да користе краћи о лик, док преусмеравања преко нацрта морају да користе потпуно квалификовано име крајње тачке које укључује име нацрта.

Да и се довршиле промене у пакету апликације, о јекти о расца се такође чувају унутар нацрта у модулу апп/маин/формс.пи.

## Апликацион Скрипт

Модул фласки.пи у директоријуму највишег нивоа је место где је инстанца апликације дефинисана. Ова скрипта је приказана у [Примеру 7-8.](#)

Пример 7-8. фласки.пи: главна скрипта

```
импорт ос
из апликације импорт цреате_апп,
д из апп.моделс импорт Усер, улога
из фласк_миграте импорт Миграте

апп = цреате_апп(ос.гетенв('ФЛАСК_ЦОНФИГ') или 'подразумевано')
миграте = Миграте(апп, д )

@app.схелл_цонтект_процессор
деф make_схелл_цонтект(): рeturн
    дицт(д =д , Усер=Усер, Роле=Роле)
```

Скрипта почиње креирањем апликације. Конфигурација се узима из променљиве окружења ФЛАСК\_ЦОНФИГ ако је дефинисана, или се у супротном користи подразумевана конфигурација. Флакс-Миграте и прилагођени контекст за Питхон љуску се затим иницијализују.

Пошто је главна скрипта апликације промењена из хелло.пи у фласки.пи, променљива окружења ФЛАСК\_АПП трећа да се ажурира у складу са тим како је команда фласк могла да лоцира инстанцу апликације. Такође је корисно омогућити Флакс-ов режим за отклањање грешака постављањем ФЛАСК\_ДЕБУГ=1. За Линук и маџОС, све се то ради на следећи начин:

```
(венв) $ екпорт ФЛАСК_АПП=фласки.пи
(венв) $ екпорт ФЛАСК_ДЕБУГ=1
```

И за Мицрософт Виндовс:

```
(венв) $ сет ФЛАСК_АПП=фласки.пи
(венв) $ сет ФЛАСК_ДЕБУГ=1
```

## Рекуирментс Филе

Доје практика да апликације укључују датотеку рекуирментс.ткт која садржи све зависности пакета, са тачним појевима верзије. Ово је важно у случају да виртуелно окружење треба да се регенерише на другој машини, као што је машина на којој ће апликација имати распоређена за производну употребу. Ову датотеку може аутоматски генерисати pip са следећом командом:

```
(венв) $ pip замрзавање > рекуирментс.ткт
```

Доје идеја да освежите ову датотеку кад год се пакет инсталира или надогради. Овде је приказан пример датотеке са захтевима:

```

алем иц==0.9.3
линкер==1.4
клика==6.7
доминате==2.3.1
Флак==0.12.2 Флак-
Боотстррап==3.3.7.1 Флак-
Майл==0.9.1 Флак-Миграте==2.0.4
Флак-Момент==0.5.1 Флак-
СКЛАлчеми==2.2 Флак-
ВТФ==0.14.2 итсдангероус==0.24
Илья2==2.9.6 Мако==1.0.7
МаркунСафе==1.0 питхон-
датеутил==2.6.1 питхон-
едитор==1.0.3

```

```

шест==1.10.0
СКЛАлчеми==1.1.11
виситор==0.1.3
Веркзеуг==0.12.2
ВТФормс==2.1

```

Када трећа да направите савршenu реплику виртуелнog окружењa, можете креirati ново виртуелно окружењe и покренути следећу команду на њему:

```
(венв) $ pip инсталл -р рекуиментс.ткт
```

Бројеви верзија у датотеци примера рекуиментс.ткт вероватно ће бити застарели до тренутка када ово прочитате. Можете покушати да користите новија издања пакета, ако желите. Ако нађете на јило какве проћлеме, увек се можете вратити на верзије наведене овде, пошто је познато да су оне компатибилне са апликацијом.

#### Јединични тестови

Ова апликација је веома мала, тако да још увек нема много тога за тестирање. Али као пример, могу се дефинисати два једноставна теста, као што је приказано у [Примеру 7-9](#).

Пример 7-9. тестс/тест\_асицс.пи: јединични тестови

```

импорт unittest из
флак импорт цуррент_апп из апликације
импорт цреате_апп, д

класс БасицсТестЦасе(unittest.ТестЦасе): деф
    сетУп(селф): селф.апп = цреате_апп('тестинг')
        селф.апп_контект = селф.апп.апп_контект()
        селф.апп_контект.пусх() д .цреате_алл()


```

```
деф тeapДовн(селф):
    д .сессион.ремове()
    д .дроп_алл()
    селф.апп_контекст.поп()
```

```
деф тест_апп_екистс(селф):
    селф.ассертFalce(цуррент_апп ис None)
```

```
деф тест_апп_ис_тестинг(селф):
    селф.ассертTrue(цуррент_апп.конфиг['ТЕСТИНГ'])
```

Тестови су написани коришћењем стандардног пакета униттест из Питхон стандардне и лиотеке. Методе сетУп() и теарДовн() класе тест цасе се покрећу пре и после сваког теста, а извршавају се све методе које имају име које почиње са тест\_ као тестови.



Ако желите да сазнате више о писању јединичних тестова са Питхон-овим униттест пакетом, прочитајте [званичну документацију](#).

СетУп() метода покушава да креира окружење за тест које је лиско окружењу апликације која ради. Прво креира апликацију конфигурисану за тестирање и активира њен контекст. Овај корак о ез еђује да тестови имају приступ цуррент\_апп, као што то чине редовни захтеви. Затим креира потпуно нову азу података за тестове користећи Флакс СКЛАлцхеми метод цреате\_алл(). База података и контекст апликације уклањају се методом теарДовн().

Први тест осигурава да инстанца апликације постоји. Други тест осигурава да апликација ради под конфигурацијом за тестирање. Да и директоријум тестова ио исправан пакет, потре но је додати модул тестс/инит.пи, или ово може ити празна датотека, пошто пакет униттест скенира све модуле да и открио тестове.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 7а да исте проверили конвертовану верзију апликације. Да исте или сигурни да имате инсталлиране све зависности, такође покрените пип инсталл -р рекуирментс.ткт.

За покретање јединичних тестова, прилагођена команда се може додати скрипти фласки.пи. Пример 7-10 показује како додати тест команду.

Пример 7-10. фласки.пи: команда покретача за тестирање јединица

```
@апп.цили.цомманд()
деф тест(): """Покрени
    тестове јединица."""
    импорт
        униттест тестс =
            униттест.ТестЛоадер().дисцовер('тестс')
        униттест.ТектТестРуннер(вер осити=2).рун (тестови)
```

Апп.цили.цомманд декоратор олакшава имплементацију прилагођених команди. Име украсене функције се користи као име команде, а низ документата функције је приказан у порукама помоћи. Имплементација функције тест() позива покретач тестира из пакета униттест .

Јединични тести се могу извршити на следећи начин:

```
(вебв) $ фласк тест
тест_апп_екистс (тест_ асицс.БасицсТестЦасе) ...
тест_апп_ис_тестинг (тест_ асицс.БасицсТестЦасе) ...

-----
О ављена 2 теста за 0,001с
```

Подешавање азе података

Реструктурирана апликација користи другачију азу података од верзије са једном скриптом.

УРЛ азе података је узет из променљиве окружења као први из ор, са подразумеваном СКЛите азом података као алтернативом. Променљиве окружења и имена датотека СКЛите азе података се разликују за сваку од три конфигурације. На пример, у развојној конфигурацији УРЛ се до ија из променљиве окружења ДЕВ\_ДАТАБАСЕ\_УРЛ, а ако то није дефинисано онда се користи СКЛите аза података са именом дата-дев.склите.

Без о зира на извор УРЛ адресе азе података, та еле азе података морају ити креирање за нову азу података. Када радите са Флакс-Миграте да исте пратили миграције, та еле азе података могу се креирати или надоградити на најновију ревизију помоћу једне нареде:

```
(вебв) $ фласк д надоградња
```

## Покретање апликације

Рефакторисање је сада завршено и апликација се може покренути. Уверите се да сте ажурирали променљиву окружења ФЛАСК\_АПП као што је наведено у „Скрипта апликације“ на страници [93](#), а затим покрените апликацију као и оночно:

```
(венив) $ flask run
```

Подешавање променљивих окружења ФЛАСК\_АПП и ФЛАСК\_ДЕБУГ сваки пут када се покрене нова сесија командне линије може да иницијализира заморно, тако да ће и трећи ало да конфигурише свој систем тако да ове променљиве буду подразумевано подешене. Ако користите асх, можете им додати у своју `~/.asrc` датотеку.

Веровали или не, стигли сте до kraја првог дела. Сада сте научили о основним елементима неопходним за прављење веб-апликације са Фласком, али вероватно нисте сигурни како се сви ови делови уклапају у праву апликацију. Циљ ИИ дела је да вам помогне у томе тако што ће вас провести кроз развој комплетне апликације.

ДЕО ИИ

---

Пример: Друштвени  
Апликација за логовање

## ГЛАВА 8

### Идентификација корисника

Већина апликација треба да прати ко су њихови корисници. Када се корисници повежу са апликацијом, они се аутентификују са њом, процес којим отварајују свој идентитет. Када апликација сазна ко је корисник, може понудити прилагођено искуство. Енце.

Најчешћи коришћени метод аутентификације захтева од корисника да остави део идентификације, што је или њихова адреса е-поште или корисничко име, и тајна која је само њима позната, а која се зове лозинка. У овом поглављу је креиран комплетан систем аутентификације за Фласки.

#### Проширења за аутентификацију за фласк

Постоји много одличних Питхон пакета за аутентификацију, али ниједан од њих не ради све. Решење за аутентификацију корисника представљено у овом поглављу користи неколико пакета и оставије лепак збор којег они дојро раде заједно. Ово је листа пакета који ће се користити и за шта се користе:

- Флакс-Логин: управљање сесијама корисника за пријављене кориснике
- Веркзеуг: хеширање и верификација лозинке
- опасно: генерисање и верификација криптографски оставије једног токена

Поред пакета специфичних за аутентификацију, користиће се следећа проширења опште намене:

- Флакс-Майл: Слање е-порука у вези са аутентификацијом
- Флакс-Боотстррап: ХТМЛ шаблони
- Флакс-ВТФ: Веб расци

## Сигурност лозинке

Без једност корисничких информација усклађиваних у азама података често се занемарује током дизајнирања већих апликација. Ако нападач успе да прорвани на ваш сервер и приступи вашој корисничкој ази података, онда ризикујете да је једност својих корисника — а ризик је већи него што мислите. Позната је чињеница да већина корисника користи исту лозинку на више сајтова, па чак и ако не чувате никакве осетљиве информације, приступ лозинкама усклађиваним у вашој ази података може да омогући нападачу приступ на лозинама које ваши корисници имају на другим сајтовима.

Кључ за је једно складиштење корисничких лозинки у ази података се ослања на то да се не чува сама лозинка, већ њен хеш. Функција хеширања лозинке узима лозинку као улаз, додаје јој насумичне компоненте (сол), а затим на њу примењује неколико једносмерних криптографских трансформација. Резултат је нови низ знакова који нема сличности са оригиналном лозинком и нема познати начин да се поново трансформише у оригиналну лозинку. Хешови лозинки се могу верификовати уместо правих лозинки јер су функције хеширања поновљиве: с овим на исте уносе (лозинку и сол), резултат је увек исти.



Хеширање лозинке је сложен задатак који је тешко исправити. Препоручује се да не имплементирате сопствено решење, већ да се ослоните на дојро познате и лиотеке које је заједница прегледала. У следећем одељку, Веркзеугове функције хеширања лозинке ће да буду демонстриране. Други дојри из ове за хеширање лозинке су [Црипт](#) и [Пассли](#). Ако сте заинтересовани да сазнате шта је укључено у генерирање је једних хешова лозинки, погледајте чланак „[Сојено хеширање лозинке – урадите то како трећи а](#)“ од Дефусе Сецурити је вредно читања.

Хеширање лозинки са Веркзеуг-овим сигурносним

модулом Веркзеуг практично имплементира је једно хеширање лозинки. Ова функционалност је изложена са само две функције, које се користе у фазама регистрације и верификације, респективно:

`генератор_парсворд_хасх(парсворд, метод='p_кдф2:сха256', салт_ленгтх=8)`

Ова функција узима лозинку у облику очичног текста и враћа хеш лозинке као стринг који се може усклађивати у корисничкој ази података. Подразумеване вредности за метод и салт\_ленгтх су довољне за већину случајева употребе.

`цхеџ_парсворд_хасх(хеш, лозинка)`

Ова функција узима хеш лозинке који је претходно усклађиван у ази података и лозинку коју је унео корисник. Повратна вредност True указује да је корисничка лозинка исправна.

**Пример 8-1** показује промене у моделу корисника креиране у **Поглављу 5** ради прилагођавања хеширања лозинке.

Пример 8-1. апп/моделс.пи: хеширање лозинке у моделу корисника

фром веркзеуг.сеџурити импорт генерате\_пассворд\_хасх, цхецк\_пассворд\_хасх

```
цласс Усер(д .Модел):
    # пассворд_хасх =
    д .Цолумн(д .Стринг(128))

    @проперти
    деф пассворд (селф):
        раписе Аттри утеЕррор('пассворд није читљив атри ут')

    @пассворд.сеттер
    деф пассворд(селф, пассворд):
        селф.пассворд_хасх = генерате_пассворд_хасх(пассворд)

    деф верифи_пассворд(селф, пассворд):
        врати цхецк_пассворд_хасх(селф.пассворд_хасх, лозинка)
```

Функција хеширања лозинке се имплементира кроз својство само за писање које се зове лозинка. Када је ово својство подешено, метод за подешавање ће позвати Веркзеугову функцију генерате\_пассворд\_хасх() и уписати резултат у поље пассворд\_хасх .

Покушај читања својства лозинке ће вратити грешку, јер се очигледно оригинална лозинка не може повратити након хеширања.

Метода верифи\_пассворд() узима лозинку и прослеђује је Веркзеуговој функцији цхецк\_пассворд\_хасх() ради верификације у односу на хеширану верзију ускладиштену у моделу корисника . Ако овај метод врати Труе, онда је лозинка исправна.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 8а да исте проверили ову верзију апликације.

Функционалност хеширања лозинке је сада завршена и може се тестирати у љусци:

```
(веб) $ фласк скелл
>>> у = Усер()
у.пассворд = 'мачка'
у.пассворд Траце ацк
(последњи позив):
Файл "<конзола>", ред 1, у <модуле> Датотека
"/хоме/фласк/флакси/апп/моделс.пи", ред 24, у подизању лозинке
Аттри утЕррор('пассворд није читљив атри ут')
Аттри утЕррор: лозинка није читљив атри ут
у.пассворд_хасх
'п қдф2:сха256:50000$моХФХ1Б$еф1574909ф9ц549285е8547цад181ц5е0213цфа44а4а а437цфа44а4а а32а7ф>
фаифива а32а79'
```

Истинито

```
>>> у.верифи_пассворд('пас')
Фалсе
```

```
>>> у2 = Усер()
у2.пассворд = 'мачка'
у2.пассворд_хасх
```

```
'п қдф2:сха256:50000$Пфз0м0КУ$27 е930 7ф0е0119д38е8д8а62ф7фбे 7ц6 7а6 7ц7 7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц7ц6а6
```

О ратите пажњу на то како покушај приступа својству лозинке корисника враћа Аттри утЕррор. Такође, корисници у и у2 имају потпуно различите хешове лозинке, иако о је користе исту лозинку. Да и се осигурало да ова функционалност настави да ради у уђубности, претходни тестови који су урађени ручно могу ити написани као јединични тестови који се могу лако поновити. У [Примеру 8-2](#) приказан је нови модул унутар пакета тестова са три нова теста који користе недавне промене у моделу корисника .

Пример 8-2. тестс/тест\_усер\_модел.пи: тестови хеширања лозинке

```
импорт unittest
фром апп.моделс импорт Усер
```

```
цлас УсерМоделТестЦасе(униттест.ТестЦасе):
    деф тест_пассворд_сеттер(селф): у = Усер( пассворд
        = 'Цат') селф.ассертТруе(у.пассворд_хасх
        није Ноне)

    деф тест_но_пассворд_геттер(селф): у =
        Усер( пассворд = 'мачка') са
        селф.ассертРаисес(Аттри утЕррор):
            у.пассворд

    деф тест_пассворд_верифицациоn(селф):
        у = Корисник( пассворд =
            'мачка') селф.ассертТруе(у.верифи_пассворд('мачка'))
        селф.ассертФалсе(у.верифи_пассворд('дог'))
```

```
деф тест_пассворд_салтс_аре_рандом(селф): у =
    Корисник(пассворд='цат') у2 = Усер(пассворд='цат')
    селф.ассертТруе(у.пассворд_хасх != у2.пассворд_хасх)
```

Да исте покренули ове нове тестове јединице, користите следећу команду:

```
(вени) $ флакс тест
тест_апп_екистс (тест_ асицс.БасицсТестЦасе) ... тест_апп_ис_тестинг      У ради
(тест_ асицс.БасицсТестЦасе) ... тест_но_пассворд_геттер      У ради
(тест_усер_модел.УсерМоделТестЦасе) ... тест_пассворд_салтс_ареМоделТестЦасе) ...
тест_пассворд_салтс_ареМодел_рандом. тест_пассворд_верификацијон (тест_усер_модел.УсерМоделТестЦасе) ...
                                                У ради
                                                У ради
```

О ављено 6 тестова за 0,379 с

Можете покренути пакет за тестирање јединица на овај начин сваки пут када желите да потврдите да све ради како се очекује. Постављање атоматизације чини верификацију ове функције веома јефтином, тако да тестирање тре а често понављати, како и се осигурало да се ова функционалност не поквари у јудићности.

## Креирање нацрта за аутентификацију

Нацрти су уведени у [Поглавље 7](#) као начин да се дефинишу руте у глобалном опсегу након што је креирање апликације премештено у фаричку функцију. У овом одељку, руте које се односе на подсистем за аутентификацију корисника иће додате другом нацрту, названом аутх. Коришћење различитих нацрта за различите подсистеме апликације је одличан начин да код уде уредно организован.

Нацрт аутентификације ће ити смештен у Питхон пакету са истим именом. Конструктор пакета нацрта креира ојекат нацрта и увози руте из модула виевс.пи. Ово је приказано у [примеру 8-3](#).

**Пример 8-3. апп/аутх/\_инит\_.пи: креирање нацрта за аутентификацију**

```
из flask импорт Блуепринт
аутх = Блуепринт('аутх', __наме__)
од . увоз приказа
```

Модул апп/аутх/виевс.пи, приказан у [примеру 8-4](#), увози нацрт и дефинише руте повезане са аутентификацијом користећи свој декоратор руте . За сада је додат /логин рута, која приказује лон за чување места са истим именом.

Пример 8-4. апп/аутх/виевс.пи: руте нацрта за аутентификацију и функције прегледа

```
из flask импорт рендер_темплате из .
импорт аутх
```

```
@аутх.роуте('/логин')
дeф логин(): рeтурн
    рендер_тeмплате('аутх/логин.хтмl')
```

Имајте на уму да је датотека ша лона дата рендер\_тeмплате() смештена у аутх директоријуму. Овај директоријум мора ити креиран унутар апликације/ша лона, пошто Флакс очекује да путање ша лона уду релативне у односу на директоријум ша лона апликације. Чувањем ша лона нацрта у сопственом поддиректоријуму, не постоји ризик од судара имена са главним нацртом или ило којим другим нацртима који ће ити додати у удућности.



Нацрти се такође могу конфигурисати да имају сопствене независне директоријуме за ша лоне. Када је конфигурисано више директоријума ша лона, функција рендер\_тeмплате() прво претражује директоријум ша лона конфигурисан за апликацију, а затим претражује директоријуме ша лона дефинисане нацртима.

Нацрт аутентификације тре да уде приложен апликацији у фа ричкој функцији цреате\_апп() , као што је приказано у [Примеру 8-5](#).

Пример 8-5. апп/\_инит\_.пи: регистрација нацрта аутентификације

```
дeф цреате_апп(цонфиг_намe):
    # ...
    из .аутх увоз аутх као аутх_ луепринт
    апп.регистер_ луепринт(аутх_ луепринт, урл_префикс='аутх')

    врати апликацију
```

Аргумент урл\_префикс у регистрацији нацрта није о авеzan. Када се користе, све руте дефинисане у нацрту иће регистроване са датим префиксом, у овом случају /аутх. На пример, /логин ruta ће ити регистрована као /аутх/логин, а потпуно квалификованi УРЛ под развојним ве сервером тада постаје [хттп://лоџалхост:5000/аутх/логин](http://лоџалхост:5000/аутх/логин).



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 8 да исте проверили ову верзију апликације.

## Аутентификација корисника помоћу Флакс-Логин-а

Када се корисници пријаве у апликацију, њихово стање аутентификације мора ити за ележено у корисничкој сесији, тако да се памти док се крећу кроз различите странице.

Флакс-Логин је мало, али изузетно корисно проширење које је специјализовано за управљање овим посе ним аспектом система за аутентификацију корисника, аз везивања за одређени механизам аутентификације.

За почетак, проширење тре да уде инсталирено у виртуелном окружењу:

(венв) \$ пип инсталл флакс-логин

Припрема корисничког модела за пријаве Флакс-Логин

лиско сарађује са сопственим корисничким објектима апликације. Да исте могли да радите са корисничким моделом апликације, проширење Флакс-Логин захтева да имплементира неколико уочијених својстава и метода. Потребне ставке су приказане у [таблици 8-1](#).

Таблици 8-1. Флакс-Логин потребне ставке

Својство/метод	Опис
ис_аутентицијатед	Мора ити Тачно ако корисник има важеће акредитиве за пријаву или Фалсе у супротном.
активан	Мора ити Тачно ако је кориснику дозвољено да се пријави или Фалсе у супротном. Вредност Фалсе се може користити за онемогућене налоге.
је_анонимно	Увек мора ити Фалсе за очне кориснике и Тачно за посебан кориснички објекат који представља анонимне кориснике.
гет_ид()	Мора да врати јединствени идентификатор за корисника, кодиран као Уницоде стринг.

Ова својства и методе могу се имплементирати директно у класу модела, али као лакшу алтернативу Флакс-Логин ове ефује класу УсерМикин која има подразумеване имплементације које су прикладне за већину случајева. Ажурирани модел корисника је приказан у [Примеру 8-6](#).

Пример 8-6. апп/модел.пи: ажурирања модела корисника за подршку пријављивања корисника

фром [флакс\\_логин](#) импорт УсерМикин

цлас Усер(УсерМикин, д .Модел):

```
_та_ленаме_ = 'усер' ид =
д .Цолумн(д .Интегер, примари_кеј = Труе ) емаил
= д .Цолумн(д .Стринг(64), јединствено=Тачно, индекс=Тачно)
корисничко_име = д .Цолумн(д .Стринг(64), јединствено=Труе, индек
=Труе) пассворд_хасх = д .Цолумн(д .Стринг(128)) роле_ид =
д .Цолумн(д .Интегер, д .ФореигнКеј('ролес.ид'))
```

Имајте на уму да је додато и поље за е-пошту . У овој апликацији, корисници ће се пријавити са својим имејл адресама, јер је мање вероватно да ће их за оправити него њихова корисничка имена.

Флакс-Логин се иницијализује у фа ричкој функцији апликације, као што је приказано у [Примеру 8-7.](#)

Пример 8-7. апп/\_инит\_.пи: Иницијализација Флакс-Логин-а

фром [флакс\\_логин импорт ЛогинМанагер](#)

```
логин_манагер = ЛогинМанагер()
логин_манагер.логин_виев = 'аутх.логин'
```

деф цреате\_апп(ционфиг\_наме):

```
# ...
логин_манагер.инит_апп(апп) #
...
...
```

Атриут логин\_виев објекта ЛогинМанагер поставља крајњу тачку за страницу за пријаву. Флакс-Логин ће преусмерити на страницу за пријаву када анонимни корисник покуша да приступи заштићеној страници. Пошто је ruta за пријаву унутар нацрта, она мора имати префикс назива нацрта.

Конечно, Флакс-Логин захтева да апликација одреди функцију која ће или позvana када екстензија трећа да учита корисника из базе података према његовом идентификатору. Ова функција је приказана у [примеру 8-8.](#)

Пример 8-8. апп/модел.пи: функција учитавања корисника

од . импорт логин\_манагер

```
@логин_манагер.усер_лоадер
деф лоад_усер(усер_ид):
    ретурн Усер.куери.гет(инг(усер_ид))
```

Декоратор логин\_манагер.усер\_лоадер се користи за регистраовање функције са Флакс-Логин, који ће је позвати када трећа да преузме информације о пријављеном кориснику. Идентификатор корисника ће или прослеђен као стринг, тако да га функција конвертује у цео рој пре него што га проследи Флакс-СКЛАЛЦХЕМИ упиту који учитава корисника. Повратна вредност функције мора или кориснички ојекат или Ноне ако је идентификатор корисника неважећи или се дододила или која друга грешка.

Заштита ruta Да и се

рута заштитила тако да јој могу приступити само аутентификовани корисници, Флакс-Логин ојезе је [декоратор логин\\_рекуиред](#). Следи пример његове употребе:

```
фром flask_login импорт логин_рекуиред

@app.route('/сецрет')
@login_required деф
секрет(): ретурн 'Само
    аутентификовани корисници су дозвољени!'
```

Из овог примера можете видети да је могуће „уланчати“ вишеструке функције декоратора. Када се два или више декоратора додају функцији, сваки декоратор утиче само на оне који су испод њега, поред циљне функције. У овом примеру, функција секрет() ће ити заштићена од неовлашћених корисника са логин\_рекуиред, а затим ће резултујућа функција ити регистрована у Флакс као пута. О рнути редослед ће произвести погрешан резултат, пошто ће оригинална функција ити регистрована као пута пре него што до ије додатна својства од декоратора логин\_рекуиред.

Захваљујући декоратору логин\_рекуиред, ако овој рути приступи корисник који није аутентификован, Флакс-Логин ће пресести захтев и уместо тога послати корисника на страницу за пријаву.

Додавање о расца за пријаву

О разац за пријаву који ће ити представљен корисницима има текстуално поље за адресу е-поште, поље за лозинку, поље за потврду „запамти ме“ и дугме за слање. Класа о расца Флакс-ВТФ која дефинише овај олик приказана је у [Примеру 8-9](#).

Пример 8-9. апп/аутх/формс.пи: о разац за пријаву

```
фром flask_wtf импорт FlaskForm
из wtforms импорт СтрингФиелд, ПасвордФиелд, БоолеанФиелд, Су митФиелд из
wtforms.валидаторс импорт DataRequiered, Ленгтх, Емаил

цлас ЛогинФорм(ФлаксФорм):
    емаил = СтрингФиелд('Емаил', валидаторс=[DataRequiered(), Ленгтх(1, 64),
                                                Емаил()])
    лозинка = ПасвордФиелд('Пасворд', валидаторс=[DataRequiered()])
    Ремем ер_ме = БоолеанФиелд('Задржи ме пријављеном') су мит =
        Су митФиелд('Пријава')
```

Класа ПасвордФиелд представља елемент <инфпут> са типе="пассворд". Класа БоолеанФиелд представља поље за потврду.

Поље е-поште користи валидаторе Ленгтх() и Емаил() из ВТФормс поред DataRequiered(), како и се осигурало да корисник не само да даје вредност за ово поље, већ и да је важећа. Приликом пружања листе валидатора, ВТФормс ће их проценити наведеним редоследом, а у случају неуспеха валидације приказана порука о грешци ће ити једна од првих валидатора који нису успели.

Ша лон повезан са страницом за пријављивање се чува у аутх/логин.хтмл. Овај ша лон само трећа да прикаже о разац користећи Флакс-Боотстррапов втф.куицк\_форм() макро.

**Слика 8-1** приказује о разац за пријаву који приказује већ претраживач.

Слика 8-1. Формулар за пријављивање

Трака за навигацију у ша лону асе.хтмл користи јиња2 услов да прикаже везе „Пријава“ или „Одјава“ у зависности од стања пријављеног тренутног корисника. Услов је приказан у [Примеру 8-10](#).

Пример 8-10. апп/темплатес/ асе.хтмл: Везе на траци за навигацију за пријављивање и одјављивање

```
<ул класс="нав нав_ар-нав нав_ар-ригхт">
  {% if цуррент_усер.ис_аутхентицатед %}
    <ли>< а хреф="{{ урл_фор('аутх.логоут') }}"> Одјава </а ></ли> {% еlse
    %} <ли>< а хреф="{{ урл_фор('аутх.логин') }}">Пријава </а ></ли> {% ендиф %}
  </ул>
```

Променљива цуррент\_усер која се користи у условном је дефинисана од стране Флакс-Логин-а и аутоматски је доступна за преглед функција и ша лона. Ова променљива садржи тренутно пријављеног корисника или прокси анонимни кориснички објекат ако корисник није пријављен. Анонимни кориснички објекти имају својство ис\_аутхентицатед постављено на Фалс, тако да

израз цуррент\_усер.ис\_аутхентицатед је згодан начин да сазнате да ли је тренутни корисник пријављен.

#### Потписивање

корисника у Имплементација функције приказа логин() приказана је у [примеру 8-11](#).

Пример 8-11. ап/аутх/виевс.пи: пута за пријаву

```
фром flask импорт рендер_темплате, редиреџт, рекуест , урл_фор, фласх фром
флакс_логин импорт логин_усер фром . импорт аутх из ..моделс импорт Усер
фром .формс импорт ЛогинФорм
```

```
@аутх.роуте('/логин', методс=['ГЕТ', 'ПОСТ']) деф
логин(): форм = ЛогинФорм() иф
форм.валидате_он_су мит():

усер = Усер.куери.фильтер_ и(емайл=форм.емайл.дата).фиরст() ако
корисник није Ништа и усер.верифи_пассворд(форм.пассворд.дата):
    логин_усер(усер, форм.ремем ер_ме.дата) нект = рекуест .аргс.гет('нект')
    ако је следеће Ништа или није нект.стартсвитех(''): нект =
        урл_фор('маин.индек') ретурн редиреџт(нект) фласх('Неважеће
        корисничко име или лозинка.') ретурн . рендер_темплате('аутх/
        логин.хтмл', форм=форм)
```

Виев функција креира ојекат ЛогинФорм и користи га као једноставан ојазнак у [поглављу 4](#). Када је захтев типа ГЕТ, функција приказа само приказује шалон, који заузврат приказује ојазнак. Када се ојазнак пошаље у ПОСТ захтеву, функција валидате\_он\_су мит () Флакс-ВТФ-а потврђује променљиве ојрасца, а затим покушава да пријави корисника.

Дајсте пријавили корисника, функција почиње учитавањем корисника из ојазнака података помоћу е-поште која се налази у ојазнаку. Ако корисник са датом адресом е-поште постоји, онда се позива његов метод верифи\_пассворд() са лозинком која је такође дошла са ојрасцем. Ако је лозинка важећа, функција Флакс-Логин-а логин\_усер() се позива да сними корисника као пријављеног за корисничку сесију. Функција логин\_усер() води корисника да се пријави и опционали „запамти ме“ Булов, који је такође послат са ојрасцем. Вредност Фалсе за овај аргумент узрокује да корисничка сесија истиче када се прозор прегледача затвори, тако да ће корисник морати поново да се пријави следећи пут. Вредност Труе доводи до постављања дуготрајног колачића у прегледачу корисника, који Флакс-Логин користи за ојављивање корисничке сесије. Опциона опција конфигурације РЕМЕМБЕР\_ЦООКИЕ\_ДУРАТИОН се може користити за промену подразумеваног једногодишњег трајања колачића за памћење.

У складу са ша лоном Пост/Редиреџт/Гет о којем се говори у [поглављу 4](#), ПОСТ захтев који је поднео акредитиве за пријаву завршава се преусмеравањем, али постоје две могуће УРЛ дестинације. Ако је о разац за пријаву представљен кориснику да и се спречио неовлашћени приступ заштићеној УРЛ адреси коју је корисник желео да посети, онда ће Флакс-Логин сачувати ту оригиналну УРЛ адресу у следећем аргументу стринга упита, којем се може приступити из речника рекуест.аргс . Ако следећи аргумент стринга упита није доступан, уместо тога се издаје преусмеравање на почетну страницу. УРЛ у следећем је валидiran да и се уверио да је релативна УРЛ адреса, како и се спречило злонамерни корисник да користи овај аргумент да преусмири несуђене кориснике на другу локацију.

У случају када је адреса е-поште или лозинка коју је дао корисник неважећа, поставља се флеш порука и о разац се поново приказује да и корисник покушао поново.



На производном серверу, апликација мора ити доступна преко ез едног ХТТП-а, тако да се акредитиви за пријаву и корисничке сесије увек преносе шифроване. Без ез едног ХТТП-а, нападач може пресести осетљиве податке током транзита.

Ша лон за пријаву тре а да се ажурира да и се о разац приказао. Ове промене су приказане у [Примеру 8-12](#).

Пример 8-12. апп/темплатес/аутх/логин.хтмл: ша лон о расца за пријаву

```
{% проширује " асе.хтмл" %}
{% импорт " оотстрап/втф.хтмл" као втф %}

{%
    лоцк титле %}Флакси – Пријава{%
    енд лоцк %}

{%
    лоцк паге_цонтент %}
<див цласc="паге-хеадер">
    <x1>Пријава</x1> </див>
<див цласc="цол-мд-4">
    {{ втф.куицк_форм(форм) }} </
        див> {%
    енд лоцк %}
```

Одјава корисника

Имплементација путање одјављивања је приказана у [Примеру 8-13](#).

Пример 8-13. апп/аутх/виевс.пи: пута за одјаву

```
фром флакс_логин импорт логоут_лосер, логин_рекуиред

@аутх.роуте('/логоут')
```

```

@логин_рекуиред
деф логоут():
    логоут_усер()
    фласх('Одјављени сте.') ретурн
    редиреџт(урл_фор('маин.индекс'))

```

Да исте одјавили корисника, Флакс-Логин-ова функција логоут\_усер() се позива да уклони и ресетује корисничку сесију. Одјава се завршава фласх поруком која потврђује акцију и преусмеравањем на почетну страницу.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит чхеџкоут 8ц да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флајск д након што проверите код. Да исте или сигурни да имате инсталиране све зависности, такође покрените пип инсталл -р рекуирментс.ткт.

#### Разумевање како функционише Флакс-Логин Флакс-

Логин је прилично мала екстензија, али з ог многих покретних делова укључених у ток аутентификације, Флакс корисници често имају про лема да разумеју како екстензија функционише. Следи редослед операција које се дешавају када се корисник пријави на систем:

- Корисник иде на `http://лоцалхост:5000/аутх/логин` кликом на „Пријава“ линк. Руковалац за ову УРЛ адресу враћа ша лон о расца за пријаву.
- Корисник уноси своје корисничко име и лозинку, и притисне дугме Пошаљи. Исти руковалац се поново позива, али сада као ПОСТ захтев вместо ГЕТ. а. Руковалац потврђује акредитиве достављене уз о разац, а затим позива Флакс-Логин-ову функцију `логин_усер()` да и пријавио корисника. . Функција `логин_усер()` записује ИД корисника у корисничку сесију као а низ.
- Функција прегледа се враћа са преусмеравањем на почетну страницу.
- Прегледач прима преусмеравање и захтева почетну страницу. а. Позива се функција приказа за почетну страницу и она покреће приказивање главног ша лона Јиња2. . Током приказивања ша лона Јиња2, референца на Флакс-Логин тренутни\_усер се појављује по први пут.
- Променљива контекста цуррент\_усер још увек нема додељену вредност за овај захтев, тако да позива интерну функцију Флакс-Логин-а `_гет_усер()` да и сазнала ко је корисник.

- д. Функција `_гет_усер()` проверава да ли постоји кориснички ИД сачуван у корисничкој сесији. Ако не постоји, враћа инстанцу Флакс-Логин-овог АнонимоусУсер-а.
- Ако постоји ИД, он позива функцију коју је апликација регистровала у декоратору `усер_лоадер`, са ИД-ом као аргументом.
- е. Руковалац `усер_лоадер` апликације чита корисника из базе података и враћа га. Флакс-Логин га додељује променљивој контекста `цуррент_усер` за тренутни захтев.
- ф. Шајлон прима ново додељену вредност `цуррент_усер`.

Декоратор `логин_рекуиред` се надограђује на променљиву контекста `цуррент_усер` тако што дозвољава да се функција украшеног приказа покрене само када је израз `цуррент_усер.ис_аутхентицатед` Тачан. Функција `логоут_усер()` једноставно прише кориснички ИД из корисничке сесије.

#### Тестирање

пријављивања Да и се проверило да ли функција пријављивања функционише, почетна страница може да се ажурира да поздрави пријављеног корисника именом. Одељак шајлона који генерише поздрав је приказан у [примеру 8-14](#).

Пример 8-14. апп/темплатес/индекс.хтмл: поздрављање пријављеног корисника

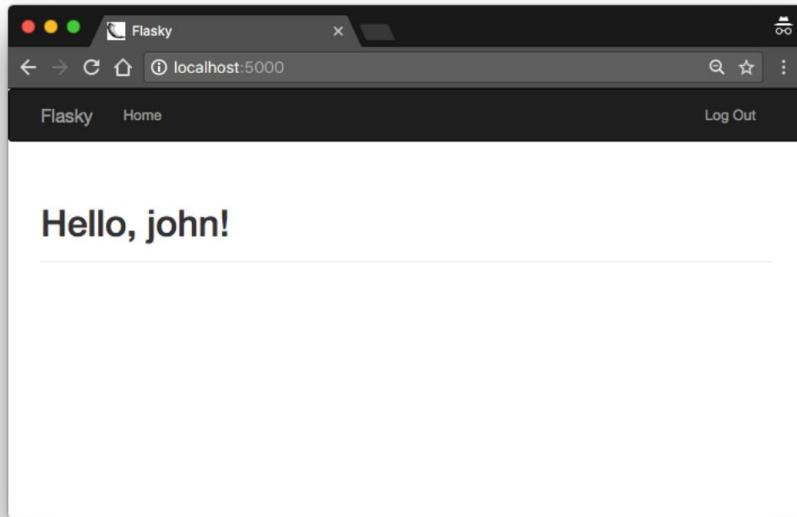
```
Здраво,  
{% иф цуррент_усер.ис_аутхентицатед %}  
  {{ цуррент_усер.усернаме }} %  
елсе %}  
  Странац  
  {% ендиф %}!
```

У овом шајлону се још једном тренутни\_усер.ис\_аутхентицатед користи за одређивање да ли је корисник пријављен.

Пошто није изграђена функција регистрације корисника, нови корисник се у овом тренутку може регистровати само из љуске:

```
(веб) $$ флакс схелл >>>  
у = Корисник(емаил='john@екампле.цом', корисничко име='john', парсворд='цат')  
>>> д .сессион.адд(у) >>> д .сессион.цоммит()
```

Претходно креирани корисник сада може да се пријави. [Слика 8-2](#) приказује почетну страницу апликације са корисником који је пријављен.



Слика 8-2. Почетна страница за успешну пријаву

## Регистрација новог корисника

Када нови корисници желе да постану чланови апликације, морају се регистровати са њом како и или познати и могли да се пријаве. Линк на страници за пријаву ће их послати на страницу за регистрацију, где могу да унесу своју адресу е-поште, корисничко име, и лозинку.

Додавање о разаца за регистрацију корисника

О разац који ће се користити на страници за регистрацију тражи од корисника да унесе адресу е-поште, корисничко име и лозинку. Овај о разац је приказан у [Примеру 8-15](#).

Пример 8-15. апп/аутх/формс.ли: о разац за регистрацију корисника

```
фром flask_втф импорт ФлаксФорм
из втформс импорт СтрингФиелд, ПассвордФиелд, БоолеанФиелд, Су митФиелд из
втформс.валидаторс импорт ДатаРекуиред, Ленгтх, Емаил, Регекп, ЕкуалТо из втформс
импорт ВалидационЕррор из ..моделс импорт Усер
```

цлас РегистратионФорм(ФлаксФорм):

```
емаил = СтрингФиелд('Емаил', валидаторс=[ДатаРекуиред(), Ленгтх(1, 64),
Емаил()])
```

```

корисничко име = СтингФиелд('Усернаме',
    валидаторс=[ ДатаРекуиред(), Ленгтх(1, 64),
        Регекп('^[А-За-з][А-За-з0-9_]*$', 0,
            'Корисничка имена морају имати само слова, ројеве,
            тачке или 'доње црте'])] лозинка = ПассвордФиелд('Пассворд',
    валидаторс=[ ДатаРекуиред(), ЕкуалТо('пассворд2', мессаже='Лозинке се морају подударати.')])
    пассворд2 = ПассвордФиелд('Потврди лозинку', валидаторс=[ДатаРекуиред()]) су мит =
Су митФиелд('Регистер')

деф валидате_емайл(селф, фиелд):
    иф Усер.куери.фильтер_ и(емайл=фиелд.дата).фирст():
        раисе ВалидатионЕррор('Имејл је већ регистрован.')

деф валидате_усернаме(селф, фиелд): иф
    Усер.куери.фильтер_ и(усернаме=фиелд.дата).фирст(): раисе
        ВалидатионЕррор('Корисничко име је већ у употребе и.')

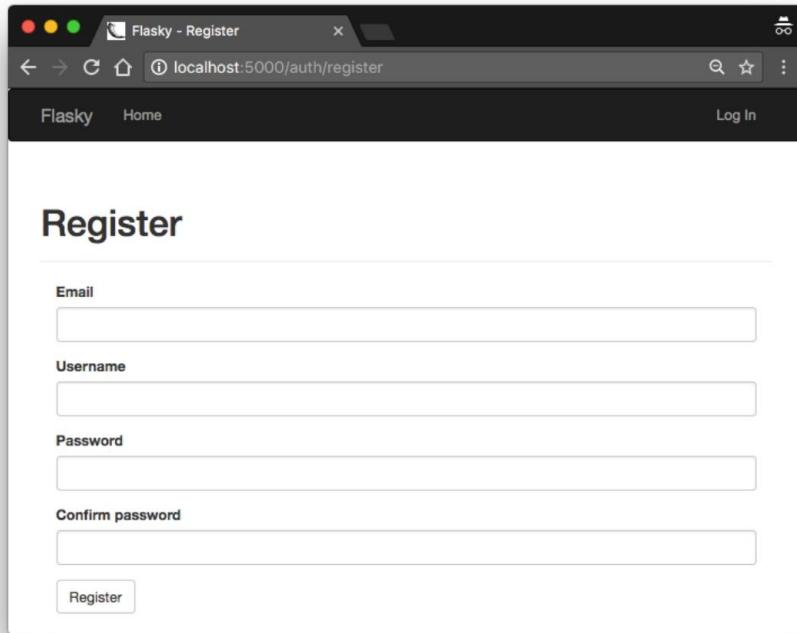
```

Овај о разац користи валидатор Регекп-а из ВТФормс како и се осигурало да поље корисничког имена почиње словом и да садржи само слова, ројеве, доње црте и тачке. Два аргумента за валидатор који прате регуларни израз су заставице регуларног израза и порука о грешци која се приказује у случају неуспеха.

Лозинка се уноси два пута као ез едносна мера, али овај корак чини неопходним да се провери да ли два поља лозинке имају исти садржај, што се ради са другим валидатором из ВТФормс који се зове ЕкуалТо. Овај валидатор је везан за једно од поља лозинке са именом другог поља датим као аргумент.

Овај о разац такође има два прилагођена валидатора имплементирана као методе. Када о разац дефинише метод са префиксом валидате\_ праћеним именом поља, метод се позива поред ило којих редовно дефинисаних валидатора. У овом случају, прилагођени валидатори за е-пошту и корисничко име осигуруја да дате вредности нису дупликати. Прилагођени валидатори указују на грешку у валидацији тако што подижу изузетак ВалидатионЕррор са текстом поруке о грешци као аргументом.

Ша лон који представља овај о разац се зове /темплатес/аутх/регистер.хтмл. Као и ша лон за пријаву, овај такође приказује о разац са втф.куиц\_форм(). Страница за регистрацију је приказана на слици 8-3.



Слика 8-3. Нови о разац за регистрацију корисника

Страница за регистрацију тре да је повезана са странице за пријаву тако да корисници који немају налог могу лако да је пронађу. Ова промена је приказана у [примеру 8-16](#).

Пример 8-16. апп/тимплатес/аутх/логин.хтмл: веза до странице за регистрацију

```
<п>
Нови корисник?
< а хреф="{{ url_for ('аутх.регистер') }}> Кликните овде
да исте се регистровали </а ></п>
```

#### Регистрација нових корисника

Руковање регистрацијама корисника не представља никаква велика изненађења. Када је о разац за регистрацију достављен и валидирање, нови корисник се додаје у азу података користећи информације које је корисник пружио. Функција приказа која оавља овај задатак приказана је у [примеру 8-17](#).

Пример 8-17. апп/аутх/виевс.пи: пута регистрације корисника

```
@аутх.роуте('/регистер', методс=[['ГЕТ', 'ПОСТ']) деф
регистер(): форм = РегистратионФорм() иф
форм.валидате_он_су мит(): корисник =
Корисник(емайл=форм.емайл.дата, корисничко
име=форм.усернаме.дата,
лозинка=форм.пассворд.дата)
д .сессион.адд(усер)
д .сессион.цоммит() фласх('Сада се можете
пријавити.') ретурн редиреџт(урл_фор('аутх/
логин')) ретурн рендер_темплате('аутх/
регистер.хтмл', форм=форм)
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхеџкоут 8д да исте проверили ову верзију апликације.

## Потврда налога

За одређене типове апликација, важно је осигурати да су подаци о кориснику дати приликом регистрације валидни. Уо ичайени захтев је да се о ез еди да се корисник може контактирати преко дате адресе е-поште.

Да и потврдиле адресу е-поште, апликације шаљу е-поруку са потврдом корисницима одмах након што се региструју. Нови налог је прво итно означен као непотврђен док се не прате упутства у мејлу, што доказује да је корисник примио е-пошту. Процедура потврде налога о ично укључује клик на посе но креiranу УРЛ везу која укључује токен за потврду.

Генерирање токена за потврду са својим опасним Најједноставнији

линк за потврду налога и ио УРЛ у формату хттп:// ввв.екампле.цом/аутх/цон рм/<ид> укључен у е-поруку за потврду, где је <ид> нумерички ИД додељена кориснику у ази података. Када корисник кликне на везу, функција прегледа која управља овом путом прима кориснички ИД за потврду као аргумент и може лако да ажурира потврђени статус корисника.

Али ово очигледно није ез една имплементација, јер ће сваки корисник који схвати формат веза за потврду моћи да потврди произвољне налоге само слањем на сумичних ројева у УРЛ-у. Идеја је да се <ид> у УРЛ-у замени токеном који садржи исте информације, али на такав начин да само сервер може да генерише важеће УРЛ-ове за потврду.

Ако се сећате дискусије о корисничким сесијама у [Поглављу 4](#), Флакс користи криптографски потписане колачиће да заштити садржај корисничких сесија од неовлашћења. Колачићи корисничке сесије садрже криптографски потпис који генерише пакет који се назива итсдангероус. Ако се садржај корисничке сесије промени, потпис се више неће подударати са садржајем, тако да Флакс од ацује сесију и започиње нову. Исти концепт се може применити на токене за потврду.

Следи кратка сесија љуске која показује како итсдангероус може да генерише потписани токен који садржи ИД корисника :

```
(венв) $ flask shell >>>
из свог опасног увоза ТимедJСОНВе СигнатуреСериализер као серијализатор >>> c =
Сериализер(апп.цонфиг['СЕЦРЕТ_КЕИ'], екпирес_ин=3600) >>> токен = с.думпс({ 'цонфирм':
23 }) >>> токен
```

```
'еиJx ГциОиИИУзИ1НиИсИмВ4цЦИ6МТМ4МТцкОду1ОЦвиавФ0ИјокМзгkНзЕ0ОТУ4фК.еи ...
>>> подаци = с.лоадс(токен) >>> подаци {'цонфирм': 23}
```

Његов опасан пакет пружа неколико типова генератора токена. Међу њима, класа ТимедJСОНВе СигнатуреСериализер генерише JСОН ве потписе (JBC) са временским истеком. Конструктор ове класе узима кључ за шифровање као аргумент, који у Флакс апликацији може или конфигурисани СЕЦРЕТ\_КЕИ.

Метода думпс() генерише криптографски потпис за податке дате као аргумент, а затим серијализује податке плус потпис као погодан низ токена.

Аргумент екпирес\_ин поставља време истека за токен, изражено у секундама.

Да и декодирао токен, ојекат серијализатора ојезије метод лоадс() који узима токен као једини аргумент. Функција проверава потпис и време истека и, ако су ојава валидна, враћа оригиналне податке. Када се методи лоадс() додели неважећи токен или важећи токен који је истекао, ствара се изузетак.

Генерирање токена и верификација помоћу ове функционалности могу се додати корисничком моделу. Промене су приказане у [примеру 8-18](#).

Пример 8-18. апп/модел.пи: потврда корисничког налога

из свог опасног увоза [ТимедJСОНВе СигнатуреСериализер](#) као серијализатор из [флакс импорт цуррент\\_апп](#) из .импорт д

[цлас Усер\(УсерМикин, д.Модел\):](#)

...

# потврђено = д.Цолумн(д.Боолеан, подразумевано=Фалсе)

```
деф генерате_цонфирматион_токен(селф, екпиратион=3600):
    с = Сериализ(циуррент_апп.ционфиг['СЕЦРЕТ_КЕИ'], екпиратион ) ретурн
    с.думлс({'цонфирм': селф.ид}).децоде('утф-8')
```

```
деф потврди (селф, токен):
    с = Сериализер(циуррент_апп.ционфиг['СЕЦРЕТ_КЕИ'])
    покушај: подаци = с.лоадс(токен.енцоде('утф-8')) осим:
        врати Фалсе
```

```
иф дана.гет('цонфирм') != селф.ид:
    ретурн Фалсе
селф.ционфирмд = Тачно
д .сессион.адд(селф)
ретурн Тачно
```

Метода генерате\_цонфирматион\_токен() генерише токен са подразумеваним временом важења од једног сата. Метода цонфирм() верификује токен и, ако је важећа, поставља нови потврђени атрибут у корисничком моделу на Тачно.

Поред верификације токена, функција цонфирм() проверава да ли се ид из токена поклапа са пријављеним корисником, који је усклађаштен у цуррент\_усер. Ово осигурује да се токен за потврду за датог корисника не може користити за потврду другог корисника.



Пошто је моделу додата нова колона за праћење потврђеног стања сваког налога, потребно је генерисати и применити нову миграцију базе података.

Две нове методе додате моделу корисника лако се тестирају у јединичним тестовима. Јединичне тестове можете пронаћи у ГитХу спремишту за апликацију.

Слање конфирмационих е-порука

Тренутна /регистер ruta преусмерава на /индек након додавања новог корисника у базу података. Пре преусмеравања, ова ruta сада треба да пошаље е-поруку са потврдом. Ова промена је приказана у примеру 8-19.

Пример 8-19. апп/аутх/виевс.пи: ruta за регистрацију са е-поштом за потврду

из ..емайл импорт сенд\_емайл

```
@аутх.роуте('/регистер', методс=['ГЕТ', 'ПОСТ']) деф
регистер(): форм = РегистрационФорм() иф
    форм.валидате_он_су мит():
```

# ...

```

д .сессион.адд(усер)
д .сессион.цоммит()
токен = усер.генерате_цонфирматион_токен()
сенд_емаил(усер.емаил, 'Потврдите свој налог', 'аутх/емаил/
    цонфирм', усер=усер, токен=токен) фласх(' Потврда е-поште
вам је послата е-поштом.') ретурн редирект(урл_фор('маин.индек')) ретурн
рендер_темплате('аутх/регистер.хтмл', форм=форм)

```

Имајте на уму да је позив д .сессион.цоммит() морао ити додат пре него што се пошаље е-порука за потврду. Про лем је у томе што се новим корисницима додељује ид када су посвећени ази података, а овај ИД је потре ан за генерирање токена за потврду.

Ша лони е-поште које користи нацрт за аутентификацију иће додати у директоријум ша лоне/аутх/емаил како и или одвојени од ХТМЛ ша лона. Као што је дискутовано у [поглављу 6](#), за сваку е-пошту су потре на два ша лона за о ичан текст и ХТМЛ верзије тела. Као пример, [Пример 8-20](#) приказује верзију са чистим текстом ша лона е-поште за потврду, а еквивалентну ХТМЛ верзију можете пронаћи у ГитХу спремишту.

Пример 8-20. апп/темплатес/аутх/емаил/цон рм.ткт: текст текста е-поште за потврду

Поштовани {{ усер.усернаме }},

До родошли у Фласки!

Да исте потврдили свој налог, кликните на следећу везу:

[{{ урл\\_фор\('аутх.цонфирм', токен=токен, \\_ектернал=True\) }}](#)

С поштовањем,

Тхе Фласки Team

Напомена: одговори на ову адресу е-поште се не прате.

Подразумевано, урл\_фор() генерише релативне УРЛ адресе; тако, на пример, урл\_фор('аутх.цонфирм', токен='а ц') враћа стринг '/аутх/цонфирм/а ц'. Ово, наравно, није важећи УРЛ који се може послати у е-поруци, јер је то само део путање УРЛ-а. Релативне УРЛ адресе функционишу до по када се користе у контексту ве странице јер их прегледач конвертује у абсолютне УРЛ-ове додавањем имена хоста и поја порта са тренутне странице, али када се УРЛ шаље путем е-поште не постоји такав контекст. Аргумент \_ектернал=True се додаје позиву урл\_фор() да се захтевала потпуно квалифицирана УРЛ адреса која укључује шему (хттп:// или хттпс://), име хоста и порт.

Функција прегледа која потврђује налоге приказана је у [примеру 8-21](#).

Пример 8-21. апп/аутх/виевс.пи: потврђивање корисничког налога

```
фром flask_логин импорт цуррент_усер
```

```
@аутх.роуте('/цонфирм/<токен>')
@логин_рекуиред деф потврди(токен):
ако тренутни_усер.цонфирмед :
    врати
        преусмеравање(урл_фор('маин.индек'))
    ако
        тренутни_усер.цонфирм (токен):
            д .сесион .цоммит() фласх(' Потврдили сте
                свој налог. Хвала!') еlse: фласх(' Веза за потврду је
                    неважећа или је истекла.') ретурн редирект(урл_фор('маин.индек'))
```

Ова ruta је заштићена декоратором логин\_рекуиред из Flask-Логин-а, тако да када корисници кликну на линк из e-поруке за потврду од њих се тражи да се пријаве пре него што дођу до ове функције прегледа.

Функција прво проверава да ли је пријављен корисник већ потврђен и у том случају преусмерава на почетну страницу, јер очигледно нема шта да се ради. Ово може да спречи непотребан рад ако корисник грешком више пута кликне на токен за потврду.

Пошто се стварна потврда токена врши у потпуности у моделу корисника , све што функција прегледа треба да уради јесте да позове методу цонфирм() и затим да трепери поруку у складу са резултатом. Када потврда успе, потврђени атрибут модела корисника се мења и додаје сесији, а затим се сесија азира података урезује.

Свака апликација може одлучити шта је дозвољено непотврђеним корисницима пре него што потврде своје налоге. Једна од могућности је да дозволите непотврђеним корисницима да се пријаве, али им покажете само страницу која од њих тражи да потврде своје налоге пре него што дођу даљи приступ.

Овај корак се може остварити коришћењем Flask-ове закачице ефоре\_рекуест , која је укратко описана у [поглављу 2](#). Из нацрта, закачивање ефоре\_рекуест се примењује само на захтеве који припадају нацрту. Да исте инсталарирали закачицу за нацрт за све захтеве апликације, уместо тога се мора користити декоратор ефоре\_апп\_рекуест . [Пример 8-22](#) показује како је имплементиран овај руководилац.

Пример 8-22. ап/аутх/виевс.пи: филтрирање непотврђених налога помоћу о рађивача ефоре\_апп\_рекуест

```
@аутх. ефоре_апп_рекуест
деф ефоре_рекуест(): ако
је цуррент_усер.ис_аутхентицатед \ а не
цуррент_усер.лонконфирмедин \ анд
рекуест.луепринт != 'аутх' \ анд
рекуест.ендпоинт != 'статици':
ретурн редиреџт(урл_фор('аутх.унлонконфирмедин'))
```

```
@аутх.роуте('/унлонконфирмедин')
деф унлонконфирмедин(): ако је
тренутни_усер.ис_анонимоус или цуррент_усер.лонконфирмедин :
ретурн редиреџт(урл_фор('маин.индек')) ретурн
рендер_темплате('аутх/унлонконфирмедин.хтмл')
```

О рађивач ефоре\_апп\_рекуест ће пресести захтев када постоје три услова

истинито:

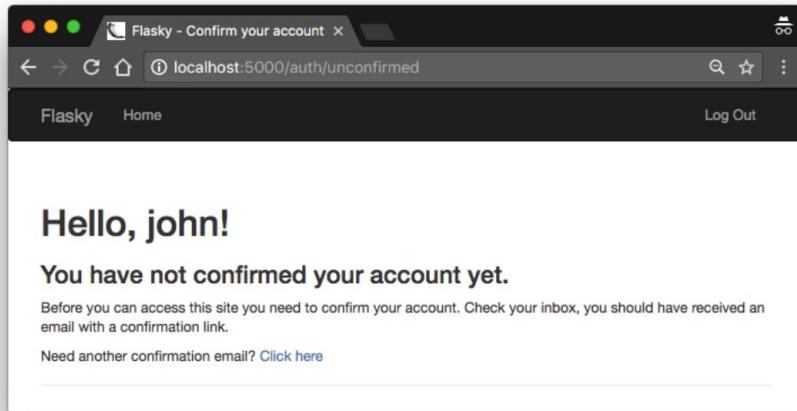
1. Корисник је пријављен (цуррент\_усер.ис\_аутхентицатед је Тачно).
2. Налог за корисника није потврђен.
3. Тражени УРЛ је изван нацрта за аутентификацију и није за статичку датотеку. Приступ путама за аутентификацију тре да уде одо рен, јер су то руте које ће омогућити кориснику да потврди налог или изврши друге функције управљања налогом.

Ако су ова три услова испуњена, онда се издаје преусмеравање на нову /аутх/ун рмед руту која приказује страницу са информацијама о потврди налога.



Када повратни позив ефоре\_рекуест или ефоре\_апп\_рекуест врати одговор или преусмерење, Флакс то шаље клијенту ез позивања функције приказа која је повезана са захтевом. Ово ефикасно омогућава овим повратним позивима да пресретну захтев када је то потребно.

Страница која се приказује непотврђеним корисницима (приказана на [слици 8-4](#)) само приказује ша лон који корисницима даје упутства како да потврде своје налоге и нуди везу за тражење нове е-поште за потврду, у случају да је оригинална е-пошта изгубљена. Рута која поново шаље е-поруку за потврду приказана је у [Примеру 8-23](#).



Слика 8-4. Непотврђена страница налога

Пример 8-23. апп/аутх/виец.пи: поновно слање е-поште за потврду налога

```
@аутх.роуте('/цонфирм')
@логин_рекуиред деф
ресенд_цонфирматион():
    токен = цуррент_усер.генерате_цонфирматион_токен()
    сенд_емайл(цуррент_усер.емайл, 'Потврдите свој налог',
        'аутх/емайл/цонфирм', усер=цуррент_усер, токен=токен)
    фласх('Нова е-пошта са потврдом вам је послата е-поштом.') ретурн
    редиреџт(урл_фор('маин.индек'))
```

Ова пута понавља оно што је урађено на рути регистрације користећи цуррент\_усер, корисника који је пријављен, као циљног корисника. Ова пута је такође заштићена са логин\_рекуиред како и се осигурало да када јој се приступи, корисник који поставља захтев је аутентификован.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 8е да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код.

## Управљање налогом

Корисници који имају налоге са апликацијом ће можда морати с времена на време да промене своје налоге. Следећи задаци се могу додати најрту аутентификације користећи технике представљене у овом поглављу:

### Ажурирање лозинки

Корисници који су свесни је једности можда желе да повремено мењају своје лозинке. Ово је лака функција за имплементацију, јер све док је корисник пријављен, је једно је представити о разац који тражи струју лозинку и нову лозинку да је замени.

Ова функција је имплементирана као урезивање 8f у ГитХу спремишту. Као део ове промене, веза „Одјава“ на траци за навигацију је претворена у падајући мени који садржи везе „Промени лозинку“ и „Одјави се“.

### Лозинка ресетује

Да исте изјегли закључавање корисника из апликације када зајправе своје лозинке, може се понудити опција ресетовања лозинке. Да исте спровели ресетовање лозинке на је један начин, неопходно је да користите токене сличне онима који се користе за потврду налога. Када корисник затражи ресетовање лозинке, порука е-поште са токеном за ресетовање се шаље на регистровану адресу е-поште. Корисник затим кликне на везу у е-поруци и, након што је токен верификован, приказује се о разац у који се може унети нова лозинка. Ова функција је имплементирана као урезивање 8g у ГитХу спремишту.

### Промене адресе е-поште

Корисницима се може дати могућност да промене своју регистровану адресу е-поште, али пре него што се нова адреса прихвати, она мора ити верификована путем е-поште за потврду. Да и користи ову функцију, корисник уноси нову адресу е-поште у о разац. Да исте потврдили адресу е-поште, на ту адресу се шаље токен. Када сервер прими токен назад, може да ажурира кориснички ојекат. Док сервер чека да прими токен, може да складиши нову адресу е-поште у ново поље азле података резервисано за адресе е-поште на чекању или може да сачува адресу у токену заједно са Ид-ом. Ова функција је имплементирана као урезивање 8x у ГитХу спремишту.

У следећем поглављу, кориснички подсистем Фласкија ће ити проширен коришћењем корисничких улога.

# ГЛАВА 9

## Улоге корисника

Нису сви корисници ве апликација једнаки. У већини апликација, малом проценту корисника поверена су додатна овлашћења која помажу да апликација ради несметано. Администратори су нај ољи пример, али у многим случајевима постоје и напредни корисници средњег нивоа, као што су модератори садржаја. Да и се ово применило, свим корисницима је додељена улога.

Постоји неколико начина за имплементацију улога у апликацији. Одговарајући метод у великој мери зависи од тога колико улога треба да буде подржано и колико су разрађене.

На пример, једноставној апликацији могу ити потреће не само две улоге, једна за обичне кориснике и једна за администраторе. У овом случају, поседовање логичког поља `is_administrator` у моделу корисника може ити све што је потребно. Сложенијој апликацији ће можда ити потреће не додатне улоге са различитим нивоима моби између редовних корисника и администратора. У неким апликацијама можда нема смисла ни говорити о дискретним улогама, а вместо тога давање скупа појединачних дозвола корисницима може ити прави приступ.

Имплементација корисничке улоге представљена у овом поглављу је хијерархија између дискретних улога и дозвола. Корисницима је додељена дискретна улога, али свака улога дефинише које радње дозвољава својим корисницима да обављају кроз листу дозвола.

### Репрезентација улога у бази података

Једноставна табела улога креирана је у [поглављу 5](#) као средство за демонстрирање односа један према више. [Пример 9-1](#) показује по ољшани модел са неким додацима.

Пример 9-1. апп/моделс.пи: модел азе података улога

цласс Роле(д .Модел):

```
_та _ленаме_ = 'ролес' ид
= д .Цолумн(д .Интегер, примари_кеи =Труе) name =
д .Цолумн(д .Стринг(64), единствено=Труе) дефаулт
= д .Цолумн( д .Боолеан, дефаулт=Фалсе, индек =Труе) дозволе =
д .Цолумн(д .Интегер) корисници = д .релатионсхип('Усер',
ацкреф='роле', лази='динамиц')
```

деф \_\_инит\_\_(селф, \*\*кваргс):

```
супер(Улога, селф).__инит__(**кваргс) ако је
селф.пермиссионс Ништа: селф.пермиссионс
= 0
```

Подразумевано поље је један од додатака овом моделу. Ово поље тре да уде подешено на Тачно за само једну улогу и Нетачно за све остале. Улога означена као подразумевана иће она која је додељена новим корисницима након регистрације. Пошто ће апликација претраживати та елу улога да и пронашла подразумевану, ова колона је конфигурисана да има индекс, јер ће то учинити претрагу много ржом.

Још један додатак моделу је поље дозвола , које је цело ројна вредност која на компактан начин дефинише листу дозвола за улогу. Пошто ће СКЛАЛЦХЕМИ ово поље подразумевано поставити на Ништа , додаје се конструктор класе који га поставља на 0 ако почетна вредност није наведена у аргументима конструктора.

Листа задатака за које су потре не дозволе је очигледно специфична за апликацију.  
За Фласки, листа је приказана у **та ели 9-1.**

### Та ела 9-1. Дозволе апликације

Назив задатка	Назив дозволе	Вредност дозволе
Пратите кориснике	ПРАТИТИ	1
Коментар на постове других КОМЕНТАР		2
Пишите чланке	ВРИТЕ	4
Модерирајте коментаре других МОДЕРАТЕ		8
Приступ администрацији	АДМИН	16

Предност коришћења овлашћења два за вредности дозвола је у томе што дозвољава ком иновање дозвола, дајући свакој могућој ком инијацији дозвола јединствену вредност за чување у пољу дозвола улоге . На пример, за корисничку улогу која корисницима даје дозволу да прате друге кориснике и коментаришу постове, вредност дозволе је ФОЛЛОВ + ЦОММЕНТ = 3. Ово је веома ефикасан начин за чување листе дозвола додељених свакој улоги.

Кодни приказ та еле 9-1 приказан је у примеру 9-2.

Пример 9-2. апп/моделс.пи: константе дозволе

класа дозвола:

```
ПРАТИ = 1
КОМЕНТАР = 2
ПИШИ = 4
УМЕРЕНА = 8
АДМИН = 16
```

Уз постављене константе дозволе, неколико нових метода се може додати моделу улога за управљање дозволама. Они су приказани у Примеру 9-3.

Пример 9-3. апп/моделс.пи: управљање дозволама у моделу улога

Улога класе (д .Модел):

```
# ...
```

```
деф адд_пермиссион(селф, перм):
    ако не селф.xас_пермиссион(perm):
        селф.пермиссионс += перм

деф ремове_пермиссион(селф, перм): иф
    селф.xас_пермиссион(perm):
        селф.пермиссионс -= перм

деф ресет_пермиссионс(селф):
    селф.пермиссионс = 0

деф хас_пермиссион(селф, перм): врати
    селф.пермиссионс & перм == перм
```

Методе адд\_пермиссион(), ремове\_пермиссион() и ресет\_пермиссион() користе основне аритметичке операције за ажурирање листе дозвола. Метод хас\_пермиссион() је најкомплекснији у скупу, јер се ослања на итоге и оператор & да проверите да ли ком инована вредност дозволе укључује дату основну дозволу. Можете се играти са овим методама у Питхон лјусци:

```
(веб) $ флаш скелл >>>
р = Улога(наме='Корисник') >>>
р.адд_пермиссион(Пермиссион.ФОЛЛОВ) >>>
р.адд_пермиссион(Пермиссион.ВРИТЕ) >>>
р.хас_пермиссион(Пермиссион. ПРАТИТИ)
Истинито
>>> р.хас_пермиссион(Пермиссион.АДМИН)

Нетачно >>> р.ресет_пермиссионс()
```

>>> p.xac\_permisson(Пермиссион.ФОЛЛОВ)  
Фалсе

Та ела 9-2 приказује листу корисничких улога које ће ити подржане у овој апликацији, заједно са ком инацијама дозвола које дефинишу сваку од њих.

Та ела 9-2. Улоге корисника

Улога корисника	Дозволе	Опис
Ниједан	Ниједан	Приступ апликацији само за читање. Ово се односи на непознате кориснике који нису пријављени.
Корисник	ПРАТИ, КОМЕНТАРИЈАЈ, ВРИТЕ	Основне дозволе за писање чланака и коментара и праћење других корисника. Ово је подразумевано за нове кориснике.
Модератор	ПРАТИ, КОМЕНТАРИЈАЈ, ПИШИ, УМЕРЕНО	Додаје дозволу за модерирање коментара других корисника.
Администратор	ПРАТИ, КОМЕНТАРИ, ПИШИ, УМЕРИ, АДМИН	Потпуни приступ, који укључује дозволу за промену улога других корисника.

Ручно додавање улога у азу података је дуготрајно и подложно је грешкама, тако да се вместо тога у класу улога може додати метод класе за ову сврху, као што је приказано у Примеру 9-4. Ово ће олакшати поновно креирање исправних улога и дозвола током тестирања јединица и, што је још важније, на производном серверу након што се апликација примени.

Пример 9-4. апп/моделс.пи: креирање улога у ази података

```
цласс Роле(д .Модел): #
    @стацијметход деф
    инсерт_ролес():
        ролес = { 'Корисник': [
            [Пермиссион.ФОЛЛОВ, Пермиссион.ЦОММЕНТ, Пермиссион.ВРИТЕ], 'Модератор':
                [Пермиссион.ФОЛЛОВ, Пермиссион.ЦОММЕНТ, Дозвола.ВРИТЕ,
                    Пермиссион.МОДЕРАТЕ], 'Администратор': [Пермиссион.ФОЛЛОВ,
                    Пермиссион.ЦОММЕНТ, Пермиссион.ВРИТЕ, Пермиссион.МОДЕРАТЕ,
                    Пермиссион.АДМИН],
            }
        } дефаулт_роле =
        'Корисник' за р у улогама:
            улога = Роле.куери.фильтер_ и(наме=p).фильтр() ако
            је улога Ноне: роле = Роле(наме=p)
            роле.ресет_пермиссионс() за перм у улогама[ р]:
            роле.адд_пермиссион(перм) роле.дефаулт =
            (роле.наме == дефаулт_роле)
```

```
д .сессион.адд(роле)
д .сессион.цоммит()
```

Функција инсерт\_ролес() не креира директно нове ојеке улоге. Уместо тога, покушава да пронађе постојеће улоге по имени и ажурира их. Нови ојекат улоге се креира само за улоге које већ нису у ази података. Ово је урађено да и се листа улога могла ажурирати уједињености када буде потребно извршити промене. Да исте додали нову улогу или променили додељене дозволе за улогу, промените речник улога на врху функције, а затим поново покрените функцију. Имајте на уму да улога „Анонимно“ не мора да буде представљена у ази података, јер је то улога која представља кориснике који нису познати и стога нису у ази података.

Такође имајте на уму да је инсерт\_ролес() статичан метод, посебан тип методе који не захтева креирање објекта јер се може позвати директно у класи, на пример, као Роле.инсерт\_ролес(). Статичке методе не узимају селф аргумент као методе инстанце.

## Додељивање улога

Када корисници региструју налог у апликацији, треба им доделити исправну улогу. За већину корисника, улога додељена у време регистрације је иће улога „Корисник“, јер је то улога која је означена као подразумевана. Једини изузетак је за администратора, коме треба од почетка доделити улогу „Администратор“. Овај корисник је идентификован по адреси е-поште која је сачувана у конфигурационој променљивој ФЛАСКИ\_АДМИН, тако да чим се та адреса е-поште појави у захтеву за регистрацију може да дође исправну улогу. [Пример 9-5](#) показује како се то ради у моделу Усер

конструктор.

Пример 9-5. апп/моделс.пи: дефинисање подразумеваване улоге за кориснике

```
цлас Усер(УсерМикин, д .Модел):
    ...
    # деф __инит__(селф, **кваргс):
        супер(Корисник, селф).__инит__(**кваргс)
        ако је селф.роле None:
            ако селф.емайл == цуррент_апп.цонфиг['ФЛАСКИ_АДМИН']:
                селф.роле = Роле.куери.филтер_ и(наме='Администратор').фирст()
            ако је селф.роле None:
                селф.роле = Роле.куери.филтер_ и(дефаулт=True).фирст()
    # ...
```

Кориснички конструктор прво позива конструкторе основних класа, а ако након тога ојекат нема дефинисану улогу, поставља администраторску или подразумевавану улогу у зависности од адресе е-поште.

## Верификација улоге

Да и се поједноставила имплементација улога и дозвола, моделу корисника може се додати помоћни метод који проверава да ли корисници имају дату дозволу у улози која им је додељена. Имплементација се једноставно ослања на претходно додане методе улога. Ово је приказано у [Примеру 9-6](#).

Пример 9-6. ап/модел.пи: процена да ли корисник има дату дозволу

```
фром flask_логин импорт УсерМикин, АнонимоусУсерМикин
```

```
цлас Усер(УсерМикин, д .Модел):
    # ...
```

```
деф цан(селф, перм):
    врати селф.роле није Ноне и селф.роле.хас_пермисион(perm)
```

```
деф ис_администратор(селф):
    врати селф.цан(Пермисион.АДМИН)
```

```
класа АнонимоусУсер(АнонимоусУсерМикин):
    деф цан(селф, пермисионс): ретурн Фалсе
```

```
деф ис_администратор(селф):
    врати Фалсе
```

```
логин_манагер.анонимоус_усер = АнонимоусУсер
```

Метода цан() додата моделу корисника враћа Тачно ако је тражена дозвола присутна у улози, што значи да кориснику трећа дозволити да изврши тражени задатак. Провера административних дозвола је толико уочијена да се такође примењује као самостална метода ис\_администратор().

За додатну погодност креирана је и прилагођена класа АнонимоусУсер која имплементира методе цан() и ис\_администратор(). Ово ће омогућити апликацији да слоју одно позива цуррент\_усер.цан() и цуррент\_усер.ис\_администратор() јез потреће да прво проверава да ли је корисник пријављен. Флакс-Логин-у је речено да користи прилагођеног анонимног корисника апликације постављањем његове класе у атријут логин\_манагер.анонимоус\_усер.

За случајеве у којима цела функција приказа трећа да буде доступна само корисницима са одређеним дозволама, може се користити прилагођени декоратор. [Пример 9-7](#) показује имплементацију два декоратора, један за генеричке провере дозвола и један који проверава специфично за дозволу администратора.

Пример 9-7. ап/децораторс.пи: прилагођени декоратери који проверавају корисничке дозволе

```
from functools import wraps
from flask import abort
from flask_login import current_user
from .models import permission

def permission_required(permission):
    def decorator(f):
        def wrapped(*args, **kwargs):
            if current_user.can(permission):
                return f(*args, **kwargs)
            else:
                abort(403)
        return wrapped
    return decorator

def admin_required(f):
    return permission_required(permission.ADMIN)
```

```
return current_user.is_authenticated
return decorator
```

```
def login_required(f):
    return permission_required(permission.LOGIN_REQUIRED)(f)
```

Ови декоратери су направљени уз помоћ **функцијских алата** пакет из Питхон стандардне и лиотеке и вратите одговор 403, „За рањено“ ХТТП статусни код, када тренутни корисник нема тражену дозволу. У **поглављу 3** креирање су прилагођене странице са грешком за грешке 404 и 500, тако да је сада страница за грешку 403 додата на сличан начин.

Следе два примера који показују употребу ових декоратора:

```
from .decorators import permission_required, permission_required
```

```
@main.route('/admin')
@login_required
@admin_required
def for_admins_only(): return "За администраторе!"
```

```
@main.route('/moderate')
@login_required
@permission_required(permission.MODERATE)
def for_moderators_only(): return "За модераторе коментара!"
```

Као правило, декоратор руте из Флакс-а треба да се да први када се користи више декоратора у функцији приказа. Преостале декоратере треба дати редоследом којим треба да процене када се позове функција приказа. У ова два случаја, прво треба проверити стање аутентификације корисника, пошто корисник треба да уде преусмерен на промпт за пријаву ако се утврди да није аутентификован.

Дозволе ће можда морати да се провере и из шалона, тако да класа Пермисион са свим њеним константама мора да има уде доступна. Да не исте морали да додајете тем-

Аргумент плоче у сваком позиву рендер\_темплате() , може се користити контекстуални процесор. Процесори контекста чине променљиве доступним свим ша лонима током рендеровања. Ова промена је приказана у примеру 9-8.

Пример 9-8. апп/майн/\_инит\_.пи: додавање класе Пермисион у контекст ша лона

```
@майн.апп_контекст_процессор
деф ињеџт_пермисионс():
    врати дицт(Пермисион=Дозвола)
```

Нове улоге и дозволе се могу користити у јединичним тестовима. Пример 9-9 приказује два теста. Изворни код на ГитХу -у укључује по један за сваку улогу.

Пример 9-9. тестс/тест\_усер\_модел.пи: јединични тестови за улоге и дозволе

```
класса УсерМоделТестЦасе(униттест.ТестЦасе):
    # ...
```

```
деф тест_усер_роле(селф):
    у = Корисник(емайл='john@екампле.цом', лозинка='мачка')
    селф.асерптРтре(у.цан(Пермисион.ФОЛЛОВ))
    селф.асерптРтре(у.цан(Пермисион.ЦОММЕНТ))
    селф.асерптРтре(у.цан(Пермисион.ВРИТЕ))
    селф.асерптФалсе(у.цан(Пермисион.МОДЕРАТЕ))
    селф.асерптФалсе(у.цан(Пермисион.АДМИН))

деф тест_анонимоус_усер(селф):
    = АнонимоусУсер()
    селф.асерптФалсе(у.цан(Пермисион.ФОЛЛОВ))
    селф.асерптФалсе(у.цан(Пермисион.ЦОММЕНТ))
    селф.асерптФалсе(у.цан(Пермисион.ВРИТЕ))
    селф.асерптФалсе(у.цан(Пермисион.МОДЕРАТЕ))
    селф.асерптФалсе(у.цан(Пермисион.АДМИН))
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит чхецкуот 9а да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код.

Пре него што пређете на следеће поглавље, додајте нове улоге својој развојној ази података у сесији Љуске:

```
(веб) $ флакс скелл
>>> Роле.инсерт_ролес()
>>> Роле.куери.алл()
[<Улога 'Администратор'>, <Улога 'Корисник'>, <Улога 'Модератор'>]
```

Такође је до ра идеја ажурирати листу корисника тако да сви кориснички налози који су или креирани пре него што су улоге и дозволе постојале имају додељену улогу. Можете покренути следећи код у Питхон лјусци да извршите ово ажурирање:

```
(веб) $ flaskсхелл  
>>> админ_роле = Роле.куери.фильтер_ и(наме='Администратор').firст()  
>>> дефолт_роле = Роле.куери.фильтер_ и(дефолт=True).firст()  
>>> за вас у Усер.куери.алл():  
...     ако је у.роле None:  
...         иф у.емайл == апп.цонфиг['ФЛАСКИ_АДМИН']:  
...             у.роле = админ_роле  
...         остало:  
...             у.роле = дефолт_роле  
...  
>>> д .сессион.цоммит()
```

Кориснички систем је сада прилично комплетан. Следеће поглавље ће га искористити за стварање странице корисничких профила.

## ГЛАВА 10

# Усер Про лес

У овом поглављу имплементирани су кориснички профили за Фласки. Сви друштвено освијештени сајтови својим корисницима дају страницу профила, на којој је представљен резиме учешћа корисника на већим страницама. Корисници могу да рекламирају своје присуство на већим локацијама тако што ће поделити УРЛ на својој страници профил, тако да је важно да УРЛ-ови буду кратки и лаки за памћење.

### Про ле Информатион

Дајте и странице са корисничким профилом или интересантније, неке додатне информације о корисницима могу се сачувати у већим података. У примеру 10-1 модел корисника је проширен са неколико нових поља.

Пример 10-1. апп/модел.пи: поља информација о кориснику

```
цлас Усер(УсерМикин, д .Модел):
    # ...
    наиме = д .Цолумн(д .Стринг(64))
    лоцијон = д .Цолумн(д .Стринг(64))
    аут_ме = д .Цолумн(д .Тект())
    мем_ер_синце = д .Цолумн(д .ДатеТиме(), дефаулт=датетиме.утцнов)
    ласт_сеен = д .Цолумн(д .ДатеТиме(), дефаулт=датетиме.утцнов)
```

Нова поља чувају право име корисника, локацију, самонаписану иографију, датум регистрације и датум последње посете. Пољу аут\_ме је додељен тип д .Тект(). Разлика између д .Стринг и д .Тект је у томе што је д .Тект поље променљиве дужине и као таквом није потребна максимална дужина.

Две временске ознаке дојијају подразумевану вредност тренутног времена. Имајте на уму да датетиме.утцнов недостаје () на крају. То је зато што подразумевани аргумент у д .Цолумн() може узети функцију као вредност. Сваки пут трећа а да ће подразумевана вредност

генерисан, СКЛАЛЦХЕМИ позива функцију да и је произвео. Ова подразумевана вредност је све што је потре но за управљање пољем мем ер\_синце .

Поље ласт\_сееен се такође иницијализује на тренутно време након креирања, али га тре а освежити сваки пут када корисник приступи сајту. За извршење овог ажурирања може се додати метод у класу Усер . Ово је приказано у примеру 10-2.

Пример 10-2. апп/модел.пи: освежавање времена последње посете корисника

`класс Усер(УсерМикин, д .Модел):`

`# ...`

`дeф пинг(селф):`

```
селф.ласт_сееен = датетиме.утцнов()
д .сессион.адд(селф) д .сессион.цоммит()
```

Да и датум последње посете за све кориснике ио ажуриран, метод пинг() мора ити позван сваки пут када се прими захтев корисника. Посто се о рађивач ефоре\_апп\_рекуест у најрту аутентификације покреће пре сваког захтева, то може лако да уради, као што је приказано у примеру 10-3.

Пример 10-3. апп/аутх/виевс.пи: пинговање пријављеног корисника

```
@аутх. ефоре_апп_рекуест
дeф ефоре_рекуест(): ако
    цуррент_усер.ис_аутхентицикад:
        цуррент_усер.пинг() ако није
        цуррент_усер.конфирмад \ и
            рекуест.ендпоинт \ анд
            рекуест. луепринт != 'аутх' \ анд
            рекуест.ендпоинт != 'статиц':
        ретурн редирецт(урл_фор('аутх.унконфирмад'))
```

## Усер Про ле Паге

Прављење странице профила за сваког корисника не представља нове изазове.

Пример 10-4 показује дефиницију руте.

Пример 10-4. апп/майн/виевс.пи: пута про ле странице

```
@майн.роуте('/усер/<усернаме>')
дeф усер(усернаме): усер =
    Усер.куери.фильтер_ и(усернаме=усернаме).фирст_ор_404() ретурн
    рендер_темплате('усер.хтмл', усер=усер)
```

Ова рута је додата у главни план. За корисника по имену јохн, страница профиле ће ити на адреси `http://лоцалхост:5000/усер/јохн`. Корисничко име дато у УРЛ-у се претражује у ази података и, ако се пронађе, ша лон усер.хтмл се приказује са њим као аргументом. Неважеће корисничко име послато на ову руту довешће до враћања грешке 404. Са Фласк-СКЛАЛцеми, случајеви претраге и грешке могу се лепо ком иновати у једној наред и користећи `first_op_404()` методу ојекта упита. Ша лон усер.хтмл ће морати да представи информације о кориснику, тако да прима кориснички ојекат као аргумент. Почетна верзија овог ша лона је приказана у [Примеру 10-5.](#)

Пример 10-5. апп/темплатес/усер.хтмл: ша лон профила корисника

```
{% проширује " асе.хтмл" %}  
{% лоцк титле %}Фласки - {{ усер.усернаме }}{{ енд лоцк %}}  
  
{% лоцк паге_центент %}  


{{ усер.усернаме }} {% иф  
усер.наме оп усер.лоцацион %}  


{% иф усер.наме %}{{ усер.наме }}{{ ендиф %}} {% иф  
усер.лоцацион %}  
        Са < a хреф="http://мапс.гугл.цом/?к={{ усер.лоцацион }}">  
            {{ усер.лоцацион }} </a>  
        а> {% ендиф %} </p> {%  
    ендиф %} {% иф  
цуррент_усер.ис_администратор() %}  


<п><a хреф="маилto:  
{{ усер.емаил }}">{{ усер.емаил }}</a></п> {%  
ендиф %} {% иф усер.а_оут_ме %}<п>{{ усер.а_оут_ме }}</п>{% ендиф % }  
  
<п>  
    Члан од {{ момент(усер.мем_ер_синце).формат('Л') }}.  
    Последњи пут виђен {{ момент(усер.ласт_сеен).формНов() }}.  
</п> </див> {% енд лоцк %}


```

Овај ша лон има неколико занимљивих детаља имплементације:

- Поља за име и локацију се приказују унутар једног елемента `<п>`. Јиња2 услов о`ез` еђује да се елемент `<п>` креира само када је дефинисано ар једно од поља.
- Поље локације корисника се приказује као веза ка упиту Гоогле мапа, тако да се кликом на њега отвара мапа центрирана на локацији.

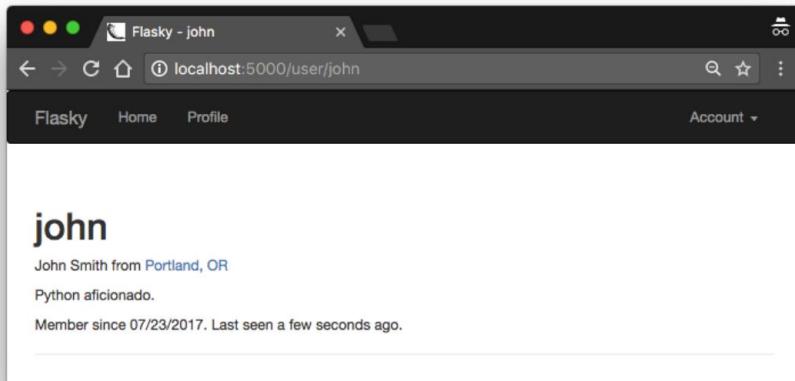
- Ако је пријављен корисник администратор, тада се приказује адреса е-поште корисника, приказана као mailto линк. Ово је корисно када администратор прегледа страницу профила другог корисника и трећа да контактира корисника.
- Две временске ознаке за корисника се приказују на страници користећи Flask-Момент, као што је приказано у [поглављу 3](#).

Пошто ће већина корисника желети лак приступ својој страници профил, веза до ње може се додати на навигациону траку. Релевантне промене ша лона асе.хтмл приказане су у [примеру 10-6](#).

Пример 10-6. аpp/темплатес/ асе.хтмл: додајте везу до профиле странице на траци за навигацију

```
{% if цуррент_усер.ис_аутхентицатед %}
<ли> < а хреф="{{ урл_фор ('майн.усер',
    усернаме= цуррент_усер.усернаме )}}">
    Профил
</а> </ли>
{% ендиф %}
```

Коришћење услова за везу странице профил је неопходно јер се трака за навигацију такође приказује за кориснике који нису аутентификовани, у ком случају се веза профил прескаче. [Слика 10-1](#) приказује како страница профил изгледа у претраживачу. Такође је приказана и нова веза профил на траци за навигацију.



Слика 10-1. Страница профил корисника



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецоут 10а да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу фласк д након што проверите код.

## Про ле Едитор

Постоје два различита случаја употребе везана за уређивање корисничких профилла. Најочигледније је да корисници морају да имају приступ страницама на којој могу да унесу информације о се и које ће представити на својим страницама профилла. Мање очигледан, али такође важан захтев је да се дозволи администраторима да уређују профиле других корисника – не само њихове личне информације, већ и друга поља у моделу корисника којима корисници немају директан приступ, као што је улога корисника. Пошто су два захтева за уређивање профилла итно различита, иће креирани два различита о расца.

Про ле Едитор на корисничком нивоу

О разац за уређивање профилла за редовне кориснике приказан је у [Примеру 10-7.](#)

Пример 10-7. апп/маин/формс.пи: уреди про ле о разац

**цласс ЕдитПрофилеФорм(ФласкФорм):**

```
наме = СтрингФиелд('Право име', валидаторс=[Ленгтх(0, 64)]) лоцацион
= СтрингФиелд('Лоцацион', валидаторс=[Ленгтх(0, 64)]) а оут_ме =
ТектАреаФиелд (' О мени') су мит = Су митФиелд('Су мит')
```

Имајте на уму да су сва поља у овом о расцу опциона, валидатор дужине дозвољава дужину од нуле као минимум. Дефиниција руте која користи овај о разац приказана је у [Примеру 10-8.](#)

Пример 10-8. апп/маин/виеовс.пи: изменни про ле руту

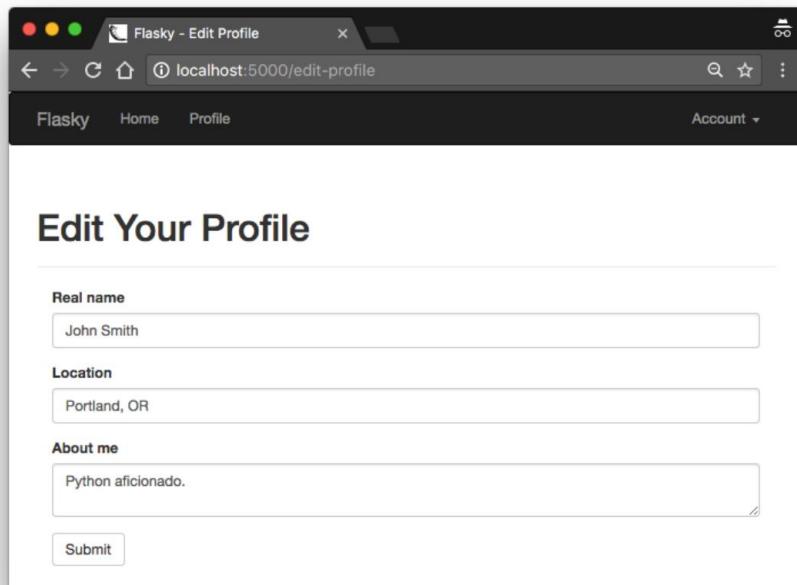
```
@маин.роуте('/едит-профиле', методс=['ГЕТ', 'ПОСТ'])
@логин_рекуире деф едит_профиле(): форм =
ЕдитПрофилеФорм() иф форм.валидате_он_су мит():
    цуррент_усер.наме = форм.наме .дата цуррент_усер.лоцацион
    = форм.лоцацион.дата цуррент_усер.а оут_ме =
        форм.а оут_ме.дата
        д .сессион.адд(цуррент_усер._гет_цуррент_о јеџт())
        д .сессион.цоммит() фласх('Ваш профил је ажуриран.')
    ретурн редиреџт(урл_фор('.усер',
        усернаме=цуррент_усер.усернаме))
```

форм.наме.дата = цуррент\_усер.наме

```
форм.лоцијон.дата = цуррент_усер.лоцијон
форм.а.оут_ме.дата = цуррент_усер.а.оут_ме
ретурн рендер_темплате('едит_профиље.хтмл', форм=форм)
```

Као иу претходним о расцима, подаци повезани са сваким пољем о расца доступни су на форм.<назив\_поља>.дата. Ово је корисно не само за до ијање вредности које је корисник доставио, већ и за пружање почетних вредности које се приказују кориснику за уређивање. Када је форм.валидате\_он\_су мит() Фалс, три поља у овом о расцу се иницијализују из одговарајућих поља у Цуррент\_усер. Затим, када се о разац пошаље, атријути података поља о расца садрже ажуриране вредности, тако да се оне премештају назад у поља корисничког објекта пре него што се ојекат сачува у бази података.

**Слика 10-2** приказује страницу за уређивање профиле.



Слика 10-2. Профил едитор

Да и корисницима олакшали да дођу до ове странице, може се додати директна веза на страницу профиле, као што је приказано у [Примеру 10-9](#).

Пример 10-9. апп/темплатес/усер.хтмл: уреди про ле линк

```
{% if усер == цуррент_усер %} < a
    класс=" тн тн-дефолт" xреф="{{ урл_фор ('.едит_профила') }}> Измени профил </ a> {%%
    endif %}
```

Услов који о ухватат везу учиниће да се веза појављује само када корисници гледају своје профиле.

### Про ле Едитор на нивоу администратора

О разац за уређивање профиле за администраторе је сложенији од оног за о ичне кориснике. Поред три поља са информацијама о профилу, овај о разац омогућава администраторима да уређују корисничку е-пошту, корисничко име, потврђен статус и улогу. О разац је приказан у примеру 10-10.

Пример 10-10. апп/майн/формс.пи: про ле о разац за уређивање за администраторе

класа [ЕдитПрофилаAdminForm\(ФлакСФорм\)](#):

```
емайл = СтингФиелд('Емайл', валидаторс=[ДатаРекуиред(), Ленгтх(1, 64),
    Емаил()])
```

```
корисничко име = СтингФиелд('Усернаме', валидаторс=[ ДатаРекуиред(),
    Ленгтх(1, 64), Регекп('^[А-За-з][А-За-з0-9_-]*$', 0,
```

```
'Корисничка имена морају имати само слова, ројеве, тачке или 'доње
црте'])]) потврђено = БолеанФиелд('Цонфирмед') роље = СелецтФиелд('Рољ',
цоерце=инт) наме = СтингФиелд('Право име', валидаторс=[Ленгтх(0, 64)]) лоцацион =
СтрингФиелд('Лоцацион', валидаторс=[Ленгтх(0, 64)]) а оут_ме = ТектАреаФиелд ('О
мени') су мит = Су митФиелд('Су мит')
```

```
деф __инит__(селф, усер, *аргс, **кваргс):
```

```
супер(ЕдитПрофилаAdminForm, селф).__инит__(*аргс, **кваргс) селф.роље.цхионес =
[(роље.ид, роље.наме) за улогу у Роле.куери.ордер_ и(Роле.наме).али()]
```

```
селф.усер = корисник
```

```
деф валидате_емайл(селф, фиелд): иф
```

```
фиелд.дата != селф.усер.емайл и \
    Усер.куери.фильтер_ и(емайл=фиелд.дата).фильтр(): раице
    ВалидатионЕррор('Е-пошта је већ регистрована.')
```

```
деф валидате_усернаме(селф, фиелд): иф
```

```
фиелд.дата != селф.усер.усернаме и \
```

```
Усер.куери.фильтер_ и(усернаме=фиелд.дата).фильтр();
приче ВалидациоnError("Корисничко име је већ у употребе и.")
```

СелеџтФиелд је омотач ВТФорм-а за контролу <селеџт> ХТМЛ о расца, која имплементира падајући листу, која се користи у овом о расцу за ода ир корисничке улоге. Инстанца СелеџтФиелд мора имати ставке постављене у атри уту из ора . Морају ити дате као листа торки, при чему се сваки тупле састоји од две вредности: идентификатора за ставку и текста који ће се приказати у контроли као стринг. Листа из ора је постављена у конструктору о расца, са вредностима до ијеним из модела улога са упитом који сортира све улоге по а ецедном реду по имени. Идентификатор за сваки тупле је постављен на ид сваке улоге, а пошто су то цели ројеви, аргумент цоерце =инт се додаје у СелеџтФиелд конструктор тако да се вредности поља чувају као цели ројеви уместо подразумеваних, а то су стрингови.

Поља е-поште и корисничког имена су конструисана на исти начин као у о расцима за аутентификацију, али њихова валидација захтева пажљиво руковање. Услов валидације који се користи за о а ова поља мора прво да провери да ли је извршена промена у пољу, а тек када дође до промене тре а да о ез еди да нова вредност не дуплира вредност другог корисника. Када се ова поља не мењају, валидација и тре ало да прође. Да и имплементира ову логику, конструктор о расца прима кориснички о јекат као аргумент и чува га као променљиву члана, која се касније користи у прилагођеним методама валидације.

Дефиниција руте за уредник профиле администратора је приказана у [Примеру 10-11](#).

Пример 10-11. апп/маин/виевс.пи: уреди руту профила за администраторе

```
из ..децораторс импорт админ_рекуиред
```

```
@маин.роуте('/едит-профиле/<инт:ид>', метод=[['ГЕТ', 'ПОСТ']] @логин_рекуиред
@админ_рекуиред деф едит_профиле_админ(ид): корисник = Усер.куери.рет_ор_404(ид)
форм = ЕдитПрофиleАдминФорм(усер=усер) иф форм.валидате_он_су мит()):
```

```
усер.емаил = форм.емаил.дата
усер.усернаме = форм.усернаме.дата
усер.конфирмед = форм.конфирмед.дата
усер.роле = Улога.куери.рет(форм.роле.дата) усер.наме =
форм.наме.дата усер.локацијон = форм.локацијон.дата
усер.а_аут_ме = форм.а_аут_ме.дата д .сесијон.адд(усер )
д .сесијон.коммит() фласх(' Профил је ажуриран.') ретурн
редиреџт(урл.фор('усер', усернаме=усер.усернаме))
```

```
форм.емаил.дата = усер.емаил
```

```

форм.усернаме.дата = усер.усернаме
форм.цонфирмд.дата = усер.цонфирмд
форм.роле.дата = усер.роле_ид
форм.наме.дата = усер.наме
форм.лоцациоn.дата = усер.лоцациоn
форм.а оут_ме.дата = усер.а оут_ме
ретурн рендер_темплате('едит_профиле.хтмл', форм=форм, усер=усер)

```

Ова ruta има углавном исту структуру као и једноставнија за о ичне кориснике, али укључује декоратор admin\_рекуриед креиран у [поглављу 9](#), који ће аутоматски вратити грешку 403 за све кориснике који нису администратори који покушају да користе ову ruta .

Ид корисника је дат као динамички аргумент у УРЛ-у, тако да се Фласк-СКЛАлцхеми-јева функција get\_op\_404() може користити, знајући да ако је ид неважећи, захтев ће вратити грешку кода 404. СелеџФиелд које се користи за улогу корисника такође заслужује да се проучи . Приликом постављања почетне вредности за поље, роле\_ид се додељује фиелд.роле.дата јер листа торки постављених у атри уту из ора користи нумеричке идентификаторе за референцу на сваку опцију. Када се о разац пошаље, ид се издваја из атри ута података поља и користи се у упиту за поновно учитавање иза раног о јекта улоге по његовом ИД -у. Аргумент цоерце =инт који се користи у декларацији СелеџФиелд у о расцу о ез јеђује да се атри ут података овог поља увек конвертује у цео рој.

За повезивање са овом страницом, на страницу корисничког профилда додаје се још једно дугме, као што је приказано у [Примеру 10-12](#).

Пример 10-12. апл/темплатес/усер.хтмл: професионална веза за уређивање за администраторе

```

{%
    if цуррент_усер.ис_администратор()
%} <a
    класс=" тн тн-дангер"
        хреf="{{ урл_фор('едит_профиле_админ', ид=усер.ид) }}> Измени
    профил [Администратор] </a> {% ендиф %}

```

Ово дугме је приказано са другачијим Боотстррап стилом да скрене пажњу на њега. Услов који га о авија чини да се дугме појављује на страницама профилда само ако пријављени корисник има улогу администратора.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 10 да исте проверили ову верзију апликације.

## Усер Аватарс

Изглед страница профиле може се по ольшати приказивањем слика аватара корисника. У овом одељку ћете научити како да додате аватаре корисника које о ез еђује Граватар, водећи сервис аватара. Граватар повезује слике аватара са адресама е-поште. Корисници креирају налог на [хттп://граватар.цом](https://gravatar.com) а затим отпремите њихове слике. Услуга излаже аватар корисника преко посе но креиране УРЛ адресе која укључује МД5 хеш адресе е-поште корисника, који се може израчунати на следећи начин:

```
(венв) $ питхон
>>> импорт хасхли
>>> хасхли .мд5('јохн@екампле.цом'.енцоде('утф-8')).хекдигест()
'д4ц74594д841139328695756648 6 дб'
```

УРЛ-ови аватара се затим генеришу додавањем МД5 хеша на хттп://сецуре.граватар.цом/аватар/УРЛ. На пример, можете да унесете хттп://сецуре.граватар.цом/аватар/д4ц74594д841139328695756648 6 дб у траку за адресу прегледача да исте до или слику аватара за адресу е-поште јохн@екампле.цом, или подразумевану слику аватара ако је та адреса е-поште нема регистрован аватар. Након што направите основну УРЛ адресу аватара, неколико аргумента стрингова упита може се користити за конфигурисање карактеристика слике аватара, као што је описано у [таблици 10-1](#).

Таблици 10-1. Граватар аргументи стрингова упита

Расправа	Опис
име	
с	Величина слике, у пикселима.
р	Оцена слике. Опције су "г", "пг", "р" и "к".
д	Подразумевани генератор слика за кориснике који немају аватаре регистроване на услуги Граватар. Опције су „404“ за враћање грешке 404, УРЛ који упућује на подразумевану слику или један од следећих генератора слика: „мм“, „идентицион“, „монстериid“, „баватар“, „ретро“ или „празно“.
фд	Присилите употребу подразумеваних аватара.

На пример, додавањем ?д=идентицион УРЛ-у аватара за јохн@екампле.цом ће се генерисати другачији подразумевани аватар који је заснован на геометријском дизајну. Све ове опције за генерисање УРЛ-ова аватара могу се додати корисничком моделу. Имплементација је приказана у [Примеру 10-13](#).

Пример 10-13. апп/моделс.пи: граватар генерисање УРЛ-а

```
импорт хасхли
из захтева за увоз очице

цлас Усер(УсерМикин, д .Модел):
    #
    деф граватар(セルф, сизе=100, дефоулт='идентицион', ратинг='г'):
```

```

урл = 'хттп://сецуре.граватар.цом/аватар' хасх =
хасхли .мд5(セルフ.емаил.ловер().енцоде('utf-8')).хекдигест() ретурн '{урл}/
{хасх }?c={сизе}&d={дефолт}&p={ратинг}'.формат(
урл=урл, хеш=хеш, величина=величина, подразумевано=подразумевано, оцена=оценка)

```

УРЛ аватара се генерише од основног УРЛ-а, МД5 хеша адресе е-поште корисника и аргумента, од којих сви имају подразумеване вредности. Имајте на уму да је један од захтева услуге Граватар да адреса е-поште са које се до ива МД5 хеш уде нормализована да садржи само мала слова а ецедних знакова, тако да се конверзија такође додаје овом методу. Са овом имплементацијом лако је генерисати УРЛ адресе аватара у Питхон љусци:

```

(венив) $ фласк скелл
>>> у = Корисник(емаил='joohn@екампле.цом')
>>> у.граватар() 'хттп://сецуре.граватар.цом/
аватар/д4ц74594д841139328695756648 6 д6?c=100&d= идентицион&p=g' >>>
у.граватар(сизе=256) 'хттп://сецуре.граватар.цом/аватар/
д4ц74594д841139328695756648 6 д6?c=256&d= идентицион&p=g'

```

Метод граватар() се такође може позвати из Јиња2 ша лона. [Пример 10-14](#) показује како се аватар од 256 пиксела може додати на страницу профила.

Пример 10-14. апп/темплатес/усер.хтмл: додавање аватара на страницу профила

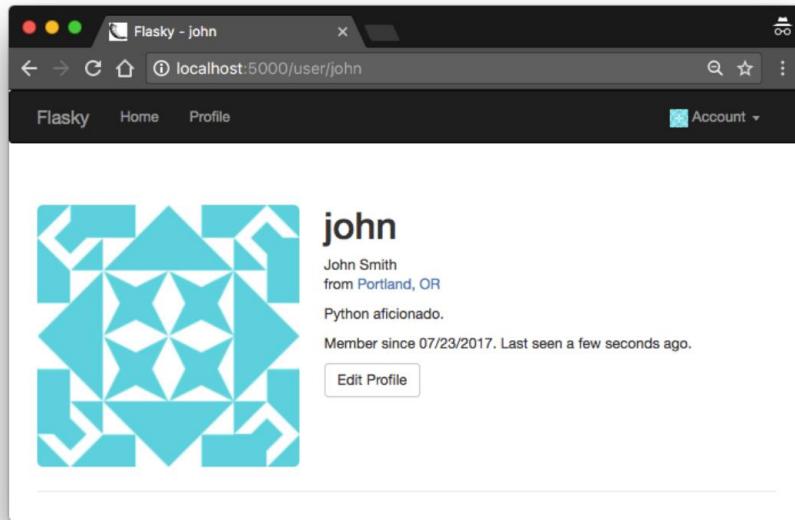
```

...
<имг цласс="имг-роундед профиле-тхум наил" срц="{{ усер.граватар(сизе=256 )}}> <див
класс="профиле-хеадер">
...
</див>
...

```

ЦСС класа за сличицу профила помаже у позиционирању слике на страници. Елемент `<див>` који прати слику о ухвата информације о профилу и користи ЦСС класу заглавља профила за по ољшање форматирања. Дефиницију ЦСС класе можете видети у ГитХу спремишту за апликацију.

Користећи сличан приступ, основни ша лон додаје малу сличицу пријављеног корисника у траку за навигацију. За оље форматирање слика аватара на страници, користе се прилагођене ЦСС класе. Можете их пронаћи у спремишту изворног кода у датотеци стилес.цсс која је дodata у фасцикли статичке датотеке апликације и на коју се упућује из ша лона асе.хтмл. [Слика 10-3](#) приказује страницу корисничког профила са аватаром.



Слика 10-3. Страница профилна корисника са аватаром



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 10ц да исте проверили ову верзију апликације.

Генерирање аватара захтева генерирање МД5 хеша, што је операција која захтева ЦПУ. Ако за страницу тре да се генерише велики рој аватара, онда се рачунски рад може зрајати и постати значајан. Пошто ће МД5 хеш за корисника остати константан све док адреса е-поште остане иста, може се кеширати у моделу корисника. [Пример 10-15](#) показује промене у моделу корисника за складиштење МД5 хешева у ази података.

[Пример 10-15. апп/модел.пи: генерирање граватар УРЛ-а са кеширањем МД5 хеша](#)

```
класс Усер(УсерМикин, д .Модел):
    # ...
    аватар_хасх = д .Цолумн(д .Стринг(32))

    деф __инит__(селф, **кваргс):
        # ...
        ако селф.емаил није Ноне и селф.аватар_хасх је Ноне:
```

```

селф.аватар_хасх = селф.граватар_хасх()

деф үханге_емаил(селф, токен):
...
# селф.емаил = нев_емаил
селф.аватар_хасх = селф.граватар_хасх()
д .сессион.адд(селф)
ретурн Тачно

деф граватар_хасх(селф):
ретурн хасхи .мд5(селф.емаил.ловер().енцоде('utf-8')).хекдигест()

деф граватар(селф, сизе=100, дефаулт='идентицион', ратинг='Г'):
иф реквест.ис_сеџре:
    урл = 'https://сеџре.граватар.цом/аватар' еlse:
        урл = 'http://ввв.граватар.цом/аватар' хасх =
            селф.аватар_хасх или селф.граватар_хасх()
ретурн '{урл}/{хасх}?с={сизе}&д={дефаулт}
&r={ратинг}'.формат
    урл=урл, хеш=хеш, величина=величина, подразумевано=подразумевано, оцена=оценка

```

Да и се из егло дуплирање логике за израчунавање граватар хеша, додат је нови метод, граватар\_хасх(), који оавља овај задатак. Током иницијализације модела, хеш се чува у новој колони модела аватар\_хасх. Ако корисник ажурира адресу е-поште, хеш се поново израчунава. Метод граватар() користи сачувани хеш ако је доступан, а ако није, генерише нови хеш као и раније.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит үхеңкоут 10д да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код.

У следећем поглављу иће креиран механизам за логовање који покреће ову апликацију.

## ГЛАВА 11

### Блог Постс

Ово поглавље је посвећено имплементацији главне функције Фласки-ја, која омогућава корисницима да читају и пишу постове на логу. Овде ћете научити неколико нових техника за поновно коришћење шаблона, пагинацију дугих листа ставки и рад са ограђеним текстом.

#### Слање и приказ ојава на логу

Да се подржали постови на логу, неопходан је нови модел азе података који их представља. Овај модел је приказан у [примеру 11-1](#).

Пример 11-1. ап/моделс.пи: Пост модел

```
цласс Пост(д .Модел):
    _та_ленаме_ = 'постс' ид
    = д .Цолумн(д .Интегер, примари_кеи =Труе)
        оди = д .Цолумн(д .Текст) тиместамп =
            д .Цолумн(д .Датетиме, индек =Труе), дефолт=датетиме.утцнов) аутхор_ид =
                д .Цолумн(д .Интегер, д .ФореигнКеи('усерс.ид'))
```

```
цласс Усер(УсерМикин, д .Модел):
    ...
    # постс = д .релатионсхип('Пост', ацкреф='аутхор', лази='динамиц')
```

Ојава на логу је представљена телом, временском ознаком и односом један-према-више из модела корисника. Поље тела је дефинисано типом д .Текст тако да нема ограничења у дужини.

О разац који ће ити приказан на главној страници апликације омогућава корисницима да напишу лог пост. Овај олик је веома једноставан; садржи само текстуалну ојаст у коју се може откуцати пост на логу и дугме за слање. Дефиниција форме је приказана у [примеру 11-2](#).

Пример 11-2. апп/маин/формс.пи: о разац за пост на логу

**цласс ПостФорм(ФлакСформ):**

оди = ТектАреаФиелд("Шта ти је на уму?", валидаторс=[ДатаРекуиред()]) су мит =  
Су митФиелд('Су мит')

Функција приказа индек() рукује формом и прослеђује листу старих постова на логу ша лону, као што је приказано у [Примеру 11-3.](#)

Пример 11-3. апп/маин/виевс.пи: ruta почетне странице са о јавом на логу

```
@маин.роуте('/', методс=['ГЕТ', 'ПОСТ']) деф
индек(): форм = ПостФорм() иф
    цуррент_усер.цан(Пермиссион.ВРИТЕ_АРТИЦЛЕС)
    и форм.валидате_он_су мит(): пост = Пост (тело=форм.тело.подаци,
        аутхор=циррент_усер._гет_циррент_о јеџт())
        д .сессион.адд(пост)
        д .сессион.цоммит()
        ретурн редиреџт(урл_фор('.индек'))
постови = Пост.куери.ордер_ и(Пост.тиместамп.десц()).алл()
ретурн рендер_темплате('индек.хтмл', форм=форм, постс=постс)
```

Ова функција прегледа прослеђује о разац и комплетну листу постова на логу у ша лон.

Листа постова је поређана према временској ознаки, у опадајућем редоследу. О разац за о јаву на логу се о рађује на уо ичайен начин, са креирањем нове инстанце поста када се прими ваљана пријава.

Дозвола тренутног корисника за писање чланака се проверава пре него што се дозволи нови пост.

О ратите пажњу на то како је атри ут аутор новог о јекта поста постављен на израз цуррент\_усер.\_гет\_циррент\_о јеџт(). Променљива цуррент\_усер из Флак Логин, као и све променљиве контекста, имплементирана је као локални проки о јекат у нити. Овај о јекат се понаша као кориснички о јекат, али је заправо танак омотач који садржи стварни кориснички о јекат унутра. Бази података је потре ан прави кориснички о јекат, који се до ија позивањем \_гет\_циррент\_о јеџт() на проки о јекту.

О разац је приказан испод поздрава у ша лону индек.хтмл, након чега следе постови на логу. Листа постова на логу је први покушај да се направи временска линија лог постова, при чему су сви постови на логу у ази података наведени хронолошким редом од најновијих до најстаријих.

Промене ша лона су приказане у [примеру 11-4.](#)

Пример 11-4. апп/темплатес/индек.хтмл: ша лон почетне странице са о јавама на логу

```

{проширује " асе.хтмл"
%} {импорт " оотстрап/втф.хтмл" као втф %}
...
<див>
{иф цуррент_усер.цан(Пермисион.ВРИТЕ_АРТИЦЛЕС)}
{{ втф.куицк_форм(форм) }} {ендиф %} </див> <ул
    класс="постс" {за постове у % } <ли класс="пост"> <див
        класс="профиле-тхум наил">

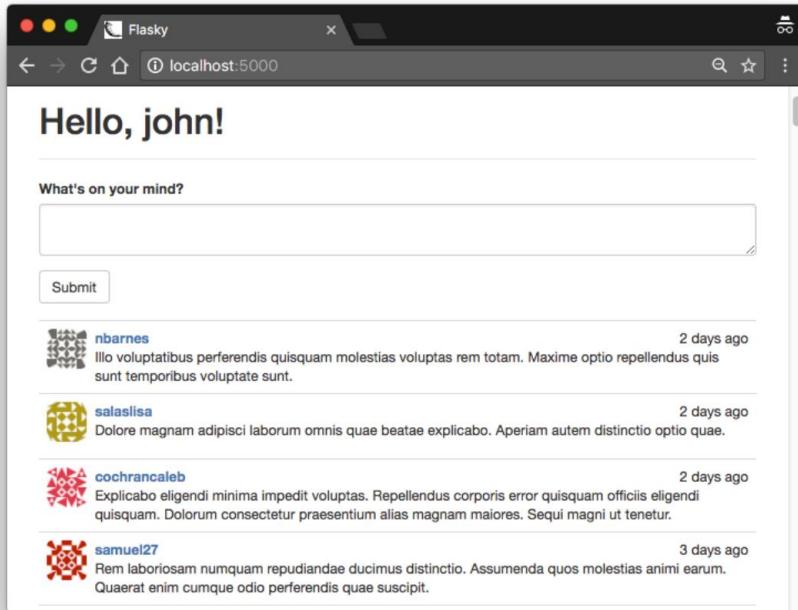
<а хреф="{{ урл_фор('. усер', усернаме=пост.аутхор.усернаме) }}">
    <имг класс="имг-роундед профиле-тхум наил"
        срц="{{ пост.аутхор.граватар(сизе=40) }}">
</а>
</див>
<див класс="пост-дате">{{ момент(пост.тиместамп).фромНов() }}</див> <див
    класс="пост-аутхор">
    <а хреф="{{ урл_фор('. усер', усернаме= пост.аутхор.усернаме) }}">
        {{ пост.аутхор.усернаме }} </а> </див> <див класс="пост-
            оди">{{ пост. оди }}</див> </ли> {ендфор %} </ул>
...

```

Имајте на уму да се метода Усер.цан() користи за прескакање о расца поста на логу за кориснике који немају дозволу ВРИТЕ у својој улози. Листа лог постова је имплементирана као ХТМЛ неуређена листа, са ЦСС класама које јој дају лепше форматирање. Мали аватар аутора приказан је на левој страни, а и аватар и корисничко име аутора су приказани као везе до странице профила корисника. Коришћени ЦСС стилови се чувају у датотеци стилес.цсс која се налази у статичком директоријуму апликације. Ову датотеку можете прегледати у ГитХу спремишту. Слика 11-1 приказује почетну страницу са о расцем за подношење и листом постова на логу.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит чхецкоут 11а да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код.



Слика 11-1. Почетна страница са формуларом за подношење лога и листом постова на логу

## Блог Постови на Про ле Пагес

Страница корисничког профила може се по ољшати приказивањем листе лог постова чији је аутор корисник. [Пример 11-5](#) показује промене у функцији приказа да и се до ила листа о јава.

Пример 11-5. апп/маин/виевс.пи: пута про ле странице са о јавама на логу

```
@майн.роуте('/усер/<усернаме>')
деф усер(усернаме): усер =
    Усер.куери.фильтер_и(усернаме=усернаме). фирмст() ако је
    корисник Ништа: а орт(404) постс =
        усер.постс .ордер_и(Пост.тиместамп.десц()).алл() ретурн
        рендер_темплате('усер.хтмл', усер=усер, постс=постс)
```

Листа лог постова за корисника се до ија из односа Усер.постс . Ово функционише као ојекат упита, тако да се филтери као што је ордер\_и() могу користити на њему као у ојичном ојекту упита.

Ша лон усер.хтмл мора да има исто <ул> ХТМЛ ста ло које приказује листу постова на логу у индек.хтмл, али потре а за одржавањем две идентичне копије дела ХТМЛ кода није идеална. За овакве случајеве, директива о укључивању Јинја2 је веома корисна.

Исечак ХТМЛ-а који генерише листу постова може да се премести у засе ну датотеку коју могу укључити и индек.хтмл и усер.хтмл. **Пример 11-6** показује како ово укључује изгледа у усер.хтмл.

Пример 11-6. апп/темплатес/усер.хтмл: про ле ша лон странице са о јавама на логу

```
...
<x3>Постови аутора {{ усер.усернаме }}</
x3> {%- укључују '_постс.хтмл' %}
...
...
```

Да и се довршила ова реорганизација, ста ло <ул> из индек.хтмл се премешта у нови ша лон \_постс.хтмл и замењује другом директивом укључивања као што је управо приказана. Имајте на уму да употребе а префикса доње црте у називу ша лона \_постс.хтмл није услов; ово је само конвенција за разликовање потпуних и делимичних ша лона.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џеџкоут 11 да исте проверили ову верзију апликације.

## Пагинирање дугих листа постова на логу

Како сајт расте и повећава се број постова на логу, постаће споро и непрактично приказивање комплетне листе постова на почетној страници и страницама profila. Великим страницама је потреба више времена за генерисање, преузимање и приказивање у већем претраживачу, тако да се квалитет корисничког искуства смањује као странице постају веће. Решење је у пагинацији података и приказивању у деловима.

## Креирање лажних података лог постова

Да исте могли да радите са више страница лог постова, неопходно је да имате тестну азу података са великим количином података. Ручно додавање нових уноса у азу података је дуготрајно и заморно; аутоматизовано решење је прикладније. Постоји неколико Питхон пакета који се могу користити за генерисање лажних информација. Прилично комплетан је Факер, који је инсталлиран са пип-ом:

```
(венив) $ pip install faker
```

Пакет Факер није, стриктно говорећи, зависност апликације, јер је потреан само током развоја. Да се одвојиле производне зависности од развојних, датотека Захтеви.ткт се може заменити поддиректоријумом Захтеви који чува различите скупове зависности. Унутар овог новог поддиректоријума, датотека дев.ткт може навести зависности које су неопходне за развој, а прод.ткт датотека може навести зависности које су потреће у производњи. Пошто постоји велики број зависности које ће ити у ове листе, за њих се додаје цоммон.ткт датотека, а затим листе дев.ткт и прод.ткт користе префикс -р да га укључиле.

**Пример 11-7** приказује датотеку дев.ткт.

Пример 11-7. Захтеви/дев.ткт: развојни захтеви ле

```
-р цоммон.ткт
факер==0.7.18
```

**Пример 11-8** показује нови модул додат апликацији који садржи две функције које генеришу лажне кориснике и објаве.

Пример 11-8. апп/факе.пи: генерирање лажних корисника и постова на логу

```
фром рандом импорт рандинт фром
склалцхеми.екц импорт ИнтегритиError фром факер
импорт Факер фром . импорт д фром .моделс импорт
Усер, Пост
```

```
деф усерс(коунт=100): факе =
    Факер() и = 0 док и <
    коунт: у =
        Усер(емайл=факе.емайл(),
            усернаме=факе.усер_наме(), парсворд=
                'парсворд', потврђено=Тачно,
                наме=лажно.име(),
                локација=лажни.град(),
                а оут_ме=лажни.текст(),
                мем ер_синце=факе.паст_дате())
        д .сессион.адд(у) покушајте:
        д .сессион.коммит()

    и += 1
```

```
осим ИнтегритиError:
    д .сессион.ролл ацк()
```

```
деф постс(коунт=100): лажни
    = Лажни () усер_коунт =
        Усер.куери.коунт()
```

```
за и у опсегу ( пој):
у = Усер.куери.оффсет(рандант(0, усер_коунт - 1)).фирст() п =
Пост( оди=факе.тект(),
      тиместамп=факе.паст_дате(),
      аутхор=у) д .сессион.адд(п)
д .сессион.цоммит()
```

Атри уте ових лажних о јеката производе на сумични генератори информација које о ез еђује Факер пакет, који може да генерише имена која изгледају стварно, имејлове, реченице и многе друге атри уте.

Адресе е-поште и корисничка имена корисника морају ити јединствена, али пошто их Факер генерише на потпуно на сумичан начин, постоји ризик од дупликата. У мало вероватном случају да се генерише дупликат, урезивање сесије азе података ће изазвати изузетак ИнтегритиЕррор. Изузетак се решава враћањем сесије уназад да и се поништио тај дуплирани корисник. Петља ће се изводити све док се не генерише тражени пој јединствених корисника.

Генерирање на сумичних постова мора свакој ојави доделити случајног корисника. За ово се користи филтер упита оффсет(). Овај филтер од ацује пој резултата датих као аргумент. Постављањем случајног померања и затим позивањем фирмст(), сваки пут се до ија другачији случајни корисник.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 11ц да исте проверили ову верзију апликације. Да исте или сигурни да имате инсталиране све зависности, такође покрените пип инсталл -р рекуирментс/дев.ткт.

Нове функције олакшавају креирање великог поја лажних корисника и постова из Питхон љуске:

```
(венв) $ фласк схелл
>>> из апликације импорт
лажно >>> факе.усерс(100)
>>> факе.постс(100)
```

Ако сада покренете апликацију, видећете дугачку листу на сумичних постова на логу на почетној страници, од много различитих корисника.

Рендеринг на страницама

**Пример 11-9** показује промене на руте почетне странице да и се подржала пагинација.

Пример 11-9. апп/майн/виевс.пи: пагинирање листе постова на логу

```
@майн.роуте('/', методс=['ГЕТ', ''ПОСТ']) деф
индек():
...
# паге = рекуест.аргс.гет('паге', 1, типе=инт)
пагинатион = Пост.куери.ордер_ и(Пост.тиместамп.десц()).пагинате( паге,
    пер_паге=циррент_апп.конфиг['ФЛАСКИ_ПОСТС_ПЕР_ПАГЕ'],
    еррор_аут=Фалсе) постови = пагинатион.итемс ретурн
рендер_темплате('индек.хтмл', форм=форм, постс=постс,
    пагинатион=пагинатион)
```

Број странице за приказ до ија се из стринга упита захтева, који је доступан као рекуест.аргс. Када страница није дата, користи се подразумевана страница од 1 (прва страница). Аргумент типе=инт осигурува да ако аргумент не може да се конвертује у цео број, враћа се подразумевана вредност.

Да и се учитала једна страница записа, последњи позив алл() методе ојекта упита се замењује са Флакс-СКЛАЛцхеми пагинате(). Метода пагинате() узима број странице као први и једини потрећни аргумент. Може се дати опционар аргумент пер\_паге да и се означила величина сваке странице, ује рој ставки. Ако овај аргумент није наведен, подразумевано је 20 ставки по страници. Други опционар аргумент који се зове еррор\_аут може ити подешен на Тачно (подразумевано) да и се издала грешка кода 404 када се тражи страница ван важећег опсега. Ако је еррор\_аут Фалсе, странице изван важећег опсега се враћају са празном листом ставки. Да и се величине страница могле конфигурирати, вредност аргумента пер\_паге се чита из конфигурационе променљиве специфичне за апликацију под називом ФЛАСКИ\_ПОСТС\_ПЕР\_ПАГЕ која се додаје у конфигурациони г.пи.

Са овим променама, листа лог постова на почетној страници ће приказати ограничен број ставки. Да и сите видели другу страницу постова, додајте стринг упита ?паге=2 у УРЛ адресу утврдила за адресу прегледача.

## Добавање виџета за пагинацију

Повратна вредност функције пагинате() је ојекат класе Пагинатион, класе коју дефинише Флакс-СКЛАЛцхеми. Овај ојекат садржи неколико својстава која су корисна за генерисање веза до страница у шајлону, тако да се прослеђује шајлону као аргумент. Резиме атријута ојекта пагинације приказан је у [табели 11-1](#).

Тајка 11-1. Атрибути објекта пагинације Флакс-СКЛАЛЦХЕМИ

Атрибут	Опис
ставке	Записи на тренутној страници
упит	Изворни упит који је пагиниран
страница	Број тренутне странице
прев_нум	Број претходне странице
нект_нум	Број следеће странице
хас_нет	Тачно ако постоји следећа страница
хас_прев	Тачно ако постоји претходна страница
странице	Укупан број страница за упит
пер_паге	Број ставки по страници
укупно	Укупан број ставки које је вратио упит

Овакат пагинације такође има неке методе, наведене у тајка 11-2.

Тајка 11-2. Методе објекта пагинације Флакс-СКЛАЛЦХЕМИ

Метод	Опис
итер_пагес(лефт_едге=2, лефт_цуррент=2, ригхт_цуррент=5, ригхт_едге=2)	Итератор који враћа низ ројева страница за приказ у пагинацији видгет. Листа ће имати леве_ивице странице на левој страни, лефт_цуррент странице лево од тренутне странице, десне_тренутне странице десно од тренутну страницу и странице десне_ивице на десној страни. На пример, за страницу 50 од 100 овај итератор конфигурисан са подразумеваним вредностима ће вратити следеће странице: 1, 2, Ништа, 48, 49, 50, 51, 52, 53, 54, 55, Ништа, 99, 100. Вредност Ништа у редослед означава празнину у редоследу страница.
прев()	Овакат пагинације за претходну страницу.
следећи()	Овакат пагинације за следећу страницу.

Наоружан овим моћним објектом и Бootstrapовим ЦСС класама пагинације, сасвим је лако направити подножје пагинације у шаплону. Имплементација приказана у

Пример 11-10 је урађен као Јиња2 макро за вишекратну употребу.

Пример 11-10. app/темплатес/\_маџрос.хтмл: макро шаплона за пагинацију

```
{% макро пагинатион_видгет(пагинација, крајња тачка) %}
<ул класс="пагинатион">
    <ли{% иф нот пагинатион.хас_прев %} класс="диса_лед"{% ендиф %}>
        <а хреф="{% иф пагинатион.хас_прев %}{% url_форендпоинт,
            страница = пагинатион.паге - 1, **кваргс %}{% еlse %}{% ендиф %}">
            &лакуо;
        </а>
    </ли>
    {% за п у пагинатион.итер_пагес() %}
```

```

{%
    иф п %
    {%
        иф п == пагинатион.паге %
        <ли класс="активе">
            < а хреф="{{ урл_фор( ендпоинт, паге = п, **кваргс ) }}{{ п }}></а> {%
                еlse %} <ли>< а хреф="{{ урл_фор(ендпоинт, паге = п, **кваргс ) }}{{ п }}></а> </
                ли> {%
                    ендиф %} {%
                    еlse %} <ли класс="диса лед">< а хреф="#">&хеллип;</а></
                    ли> {%
                        ендиф %} {%
                        ендфор %} <ли{%
                            иф нот пагинатион.хас_нект %}
                            класс="диса лед"{{ ендиф %}}> < а хреф="{{ иф пагинатион.хас_нект }}"
                            {{ урл_фор(ендпоинт,

```

страница = пагинатион.паге + 1, \*\*кваргс ) }}{%
 еlse %}#{{ ендиф %}}>

&ракуо; </а > </ли> </ул> {%
 ендмацро %}

Макро креира Боотстррап елемент пагинације, који је стилизована неуређена листа. Дефинише следеће везе до страница унутар њега:

- Линк „претходна страница“. Ова веза до ија онемогућену ЦСС класу ако је тренутна страница је прва страница.

- Везе ка свим страницама које враћа итер\_пагес() ојекта пагинације .

Ове странице се приказују као везе са експлицитним ројем странице, датим као аргумент урл\_фор(). Страница која је тренутно приказана је истакнута помоћу активне ЦСС класе.

- Празнине у редоследу страница приказане су знаком елипсе. • Веза „следећа страница“. Ова веза ће се појавити онемогућена ако је тренутна страница последња страна.

Макро Јиња2 увек примају аргументе кључне речи ез потре е да укључују \*\*кваргс у листу аргумената. Макро за пагинацију прослеђује све аргументе кључне речи које прими позиву урл\_фор() који генерише везе за пагинацију. Овај приступ се може користити са рутама као што је страница профила које имају динамички део.

Макро пагинатион\_видгет се може додати испод ша лона \_постс.хтмл укљученог у индек.хтмл и усер.хтмл. **Пример 11-11** показује како се користи на почетној страници апликације.

Пример 11-11. апп/темплатес/индек.хтмл: подноје пагинације за листе постова на логу

```
{% проширује " асе.хтмл"
%} {% увоз " оотстрап/втф.хтмл" као втф %}
{% увоз "_мацрос.хтмл" као макрое %}
...
{% инклуде '_постс.хтмл' %}
<див класс="пагинатион">
    {{ мацрос.пагинатион_видгет(пагинатион, '.индек') }} </див>
{% ендиф %}
```

Слика 11-2 показује како се везе за пагинацију појављују на страници.



Слика 11-2. Пагинација постова на логу



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит чхеџкот 11д да исте проверили ову верзију апликације.

## Постови о огађеног текста са Маркдовн и Фласк-Пагедовн

Постови у о ичном тексту су довољни за кратке поруке и ажурирања статуса, али корисници који желе да пишу дуже чланке сматраће да недостатак форматирања веома ограничава. У овом одељку, поље поља за текст у које се уносе постови иће надограђено да подржи смањење синтаксу и представити преглед о огађеног текста поста.

Имплементација ове функције захтева неколико нових пакета:

- Пагедовн, Маркдовн-то-ХТМЛ конвертор на страни клијента имплементиран у Јави-Скрипта
- Фласк-Пагедовн, омотач Пагедовн за Фласк који интегрише Пагедовн са Фласк-ВТФ о расци
- Маркдовн, конвертор Маркдовн-то-ХТМЛ на страни сервера имплементиран у Питхон
- Блеаџ, ХТМЛ санитизер имплементиран у Питхон-у

Сви Питхон пакети се могу инсталаријати помоћу пип-а:

```
(венв) $ pip install flask-pagedown markdown bleach
```

Коришћење Фласк-Пагедовн

Екstenзија Фласк-Пагедовн дефинише класу ПагедовнФиелд која има исти интерфејс као ТектАреаФиелд из ВТФормс. Пре него што се ово поље може користити, екстензију треба иницијализовати као што је приказано у [Примеру 11-12.](#)

Пример 11-12. апп/\_инит\_.пи: Фласк-Пагедовн иницијализација

```
from flask_pagedown import Pagedown
pagedown = Pagedown()

...
def create_app(config_name):
    ...
    # pagedown.init_app(app)
    # ...


```

Да исте конвертовали контролу о ласти текста на почетној страници у Маркдовн уређивач огатог текста, поље тела ПостФорм-а мора ити промењено у ПагедовнФиелд као што је приказано у [Примеру 11-13.](#)

Пример 11-13. апп/маин/формс.пи: о разац за пост са омогућеним Маркдовн

```
from flask_pagedown.fields import PagedownField
```

класс ПостФорм(ФласкФорм):

```
    оди = PagedownField("Шта ти је на уму?", validators=[Optional()])
    мит = Су митФиелд('Пошаљи')
```

Преглед Маркдовн-а се генерише уз помоћ Пагедовн и лиотека, тако да се оне морају додати у ша лон. Фласк-Пагедовн поједностављује овај задатак пружањем

узимање ша лонског макроа који укључује потре не датотеке са ЦДН-а као што је приказано у Примеру 11-14.

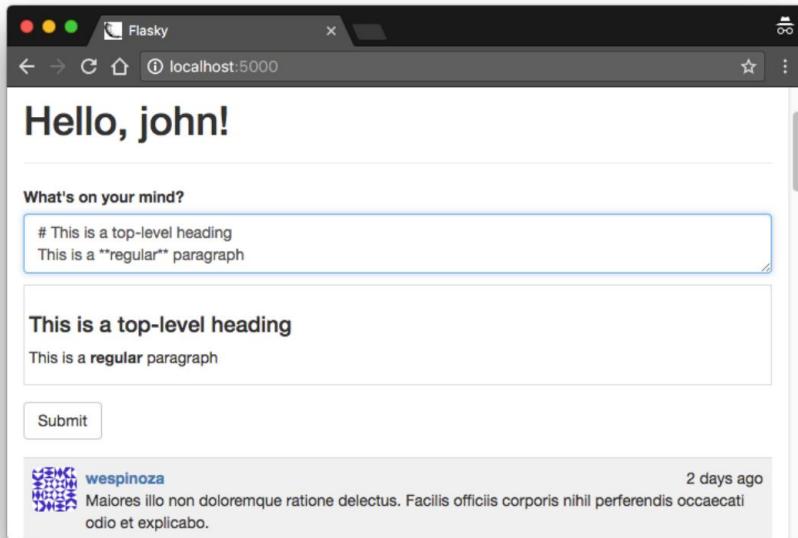
Пример 11-14. апп/темплатес/индек.хтмл: декларација ша лона Фласк-Пагедовн

```
{% лок скрипти %}
{{ супер() }}
{{ пагедовн.инклуде_пагедовн() }} {%
енд лоцк %}
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкуот 11е да исте проверили ову верзију апликације. Да исте или сигурни да имате инсталiranе све зависности, такође покрените пип инсталл -р рекуирментс/дев.ткт.

Са овим променама, текст форматиран у Маркдовн-у укуцан у поље за текст ће се одмах приказати као ХТМЛ у о ласти за преглед испод. Слика 11-3 приказује о разац за подношење лога са о огађеним текстом.



Слика 11-3. О разац за лог са о огађеним текстом

Руковање о огађеним текстом на

серверу Када се о разац пошаље, са ПОСТ захтевом се шаље само сирови Маркдован текст ; ХТМЛ преглед који је приказан на страници се од ацује. Слање генерисаног ХТМЛ прегледа са формом може се сматрати ез едносним ризиком, јер и нападачу ил о прилично лако да направи ХТМЛ секвенце које се не поклапају са Маркдован извором и пошаље их. Да исте из егли ил о какве ризике, шаље се само Маркдован изворни текст и када се на серверу поново конвертује у ХТМЛ користећи Маркдован, Питхон Маркдован-то-ХТМЛ претварач. До ијени ХТМЛ се дезинфира помоћу Блеацах-а како и се осигурало да се користи само кратка листа дозвољених ХТМЛ ознака.

Конверзија постова на логу Маркдован у ХТМЛ може се о авити у ша лону \_постс.хтмл, али то је неефикасно јер ће постови морати да се конвертују сваки пут када уду приказани на страници. Да и се из егло ово понављање, конверзија се може извршити једном када се лог пост креира, а затим кешује у ази података. ХТМЛ код за приказани лог пост се кешује у ново поље које је додато у Пост модел коме ша лон може директно да приступи. Оригинални Маркдован извор се такође чува у ази података у случају да о јаву тре а уредити. **Пример 11-15** показује промене у моделу Пост

Пример 11-15. апп/моделс.пи: Руковање текстом Маркдован у моделу Пост

из маркдован увоз маркдован увоз  
из ељивач

```
класс Пост(д .Модел):
    # ..ОДИ_ХТМЛ =
    д .Цолумн(д .Текст) #
    ...
    @статицметод
    деф он_цхангед_ оди(циль, вредност, стара вредност, покретач):
        дозвољено_tags = ['a', 'a    p', 'акроним', ' ', ' локцитат', 'код', 'ем', 'и', 'ли',
        'ол', 'пре', 'јако', 'ул', 'x1', 'x2', 'x3', 'п']
        ...
        таргет. оди_хтмл =
            леацх.линкифи( леацх.цлеан( маркдован(валуе,
            оутпут_формат='хтмл'), тагс=алловед_тагс, стрип=True))
        д .евент.листен(Пост. оди, 'сет', Пост.он_цхангед_ оди)
```

Функција он\_цхангед\_ оди() је регистрована као слушалац СКЛАлцхеми-јевог „сет“ догађаја за тело, што значи да ће ити аутоматски позвана кад год се поље тела постави на нову вредност. Функција руковаца приказује ХТМЛ верзију тела и складишти је у оди\_хтмл, ефективно чинећи конверзију Маркдован текста у ХТМЛ потпуно аутоматском.

Стварна конверзија се врши у три корака. Прво, функција маркдован() врши почетну конверзију у ХТМЛ. Резултат се прослеђује цлеан(), заједно са листом

одо рене ХТМЛ ознаке. Цлеан () функција уклања све ознаке које нису на елој листи. Коначна конверзија се врши помоћу линкифи(), још једне функције коју о ез еђује Блеацх и која претвара све УРЛ-ове написане у о ичном тексту у исправне <a> везе. Овај последњи корак је неопходан јер аутоматско генерирање везе није званично у Маркдовн спецификацији, али је веома згодна карактеристика. На страни клијента, Пагедовн подржава ову функцију као опционалну екстензију, тако да линкифи() одговара тој функцији на сервер.

Последња промена је замена пост. оди са пост. оди\_хтмл у ша лону када је доступан, као што је приказано у [Примеру 11-16](#).

Пример 11-16. апп/темплатес/\_постс.хтмл: користите ХТМЛ верзију тела постова у ша лону

```
...
<див цлас="пост- оди">
  {%- if пост. оди_хтмл %}
    {{ пост. оди_хтмл | сигурно }}
  {% остало %} {{ пост. оди }} {%
    ендиф %} </див>
```

...

Тхе | сигуран суфикс када се приказује ХТМЛ тело је ту да каже Јиња2 да не из егава ХТМЛ елементе. Јиња2 подразумевано из егава све променљиве ша лона као ез едносну меру, али ХТМЛ који је генерирао Маркдовн је генерирао сервер, тако да је ез едно да се прикаже директно као ХТМЛ.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкот 11ф да исте проверили ову верзију апликације. Ово ажурирање такође садржи миграцију азе података, тако да не за оправите да покренете фласк д упграде након што проверите код. Да исте или сигурни да имате инсталлиране све зависности, такође покрените пип инсталл -р рекуирментс/дев.ткт.

## Сталне везе до постова на логу

Корисници ће можда желети да поделе везе до одређених постова на логу са пријатељима на друштвеним мрежама. У ту сврху, сваком посту ће ити додељена страница са јединственим УРЛ-ом који га упућује. Функције руте и прегледа које подржавају сталне везе приказане су у [Примеру 11-17](#).

Пример 11-17. апп/маин/виевс.пи: омогућавање сталних веза до постова

```
@маин.роуте('/пост/<инт:ид>') деф
пост(ид): пост =
    Пост.куери.get_op_404(ид) ретурн
    рендер_темплате('пост.хтмл', постс=[пост])
```

УРЛ-ови који ће ити додељени о јавама на логу се конструишу са јединственим пољем ид који се додељује када се о јава у аци у азу података.



За неке врсте апликација, прављење трајних веза које користе читљиве УРЛ адресе уместо нумеричких ИД-ова може ити пожељније. Алтернатива нумеричким ИД-овима је да се сваком посту на логу додели пуж, који је јединствени низ који се заснива на наслову или првих неколико речи поста.

Имајте на уму да ша лон пост.хтмл прима листу са једним елементом који представља о јаву за приказивање. Слање листе је ствар погодности, тако да се ша лон \_постс.хтмл на који упућују индек.хтмл и усер.хтмл може користити и на овој страници.

Сталне везе се додају на дно сваке о јаве у генеричком ша лону \_постс.хтмл, као што је приказано у [Примеру 11-18](#).

Пример 11-18. апп/темплатес/\_постс.хтмл: додање трајних веза до постова

```
<ул класс="постс">
    {% за пост у постовима %}
        <ли класс="пост">
            ...
            <див класс="пост-центент">
                ...
                <див класс="пост-фоотер">
                    <a хреф="{{ урл_фор ('пост', ид=пост.ид) }}>
                        <спан класс="ла ел ла ел-дефаулт">Пермалинк</спан>
                    </а> </див> </див> </ли> {% ендфор %} </ул>
```

Нови пост.хтмл ша лон који приказује страницу са трајним линком приказан је у [Примеру 11-19](#). Укључује пример ша лона.

Пример 11-19. апп/темплатес/пост.хтмл: ша лон трајног линка

```
{% проширује " асе.хтмл" %}

{% лоцк титле %}Фласки - Пост{% енд лоцк %}

{% лоцк паге_цонтент %}
{% инклуде '_постс.хтмл' %} {% 
енд лоцк %}
```



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 11г да исте проверили ову верзију апликације.

## Уредник логова

Последња функција везана за постове на логу је уређивач постова који омогућава корисницима да уређују своје постове. Уређивач постова на логу живи на самосталној страници и такође је заснован на Флакс Пагедовн, тако да о ласт текста у којој се може уређивати Маркдован текст лог поста прати рендеровани преглед. Ша лон едит\_пост.хтмл приказан је у [примеру 11-20](#).

Пример 11-20. апп/темплатес/едит\_пост.хтмл: уређивање ша лона поста на логу

```
{% проширује " асе.хтмл" %}
{% импорт " оотстрап/втф.хтмл" као втф %}

{% лоцк титле %}Фласки - Уреди пост{% енд лоцк %}

{% лоцк паге_цонтент %}
<див цласс="паге-хеадер">
  <x1>Измени пост</x1> </
див> <див>
  {{ втф.куицк_форм(форм) }} </
  див> {% енд лоцк % }
```

```
{% лок скрипти %}
{{ супер() }}
{{ пагедовн.инклуде_пагедовн() }} {% 
енд лоцк %}
```

Рута која подржава уређивач постова на логу приказана је у [примеру 11-21](#).

Пример 11-21. апп/майн/виевс.пи: уреди руту поста на логу

```
@маин.роуте('/едит/<инт:ид>', методс=['ГЕТ', 'ПОСТ'])
@логин_рекуиред деф едит(ид): пост = Пост.куери.гет_оп_404(ид)
иf цуррент_усер != пост.аутхор и \ не
    цуррент_усер.цан(Пермиссион.АДМИН): а орт(403) форм
        = ПостФорм() ако форм.валидате_он_су мит():

    пост. оди = форм. оди.дата
    д .сессион.адд(пост)
    д .сессион.цоммит() фласх()
    О јава је ажурирана.') ретурн
        редирект(урл_фор('пост', ид=пост.ид) )
    форм. оди.дата = пост. оди ретурн
    рендер_темплате('едит_пост.хтмл', форм=форм)
```

Ова функција приказа је кодирана тако да дозвољава само атору лог поста да га уређује, осим администраторима, којима је дозвољено да уређују постове свих корисника. Ако корисник покуша да измене о јаву другог корисника, функција прегледа одговара кодом 403. Класа ве о расца ПостФорм која се користи овде је иста она која се користи на почетној страници.

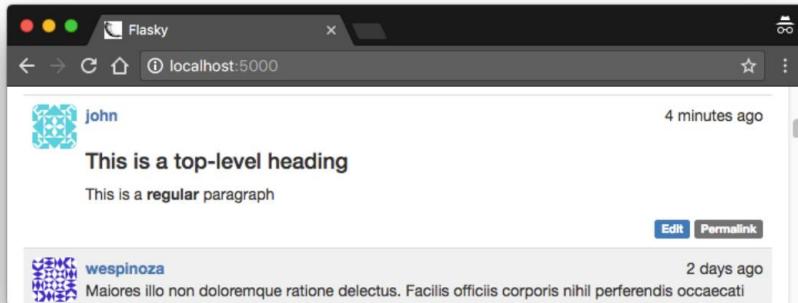
Да исте довршили ову функцију, линк ка уређивачу лог постова се може додати испод сваког поста на логу, поред трајне везе, као што је приказано у Примеру 11-22.

Пример 11-22. апп/темплатес/\_постс.хтмл: додавање везе за уређивање поста на логу

```
<ул класс="постс">
    {%- за пост у постовима %}
    <ли класс="пост">
        ...
        <див класс="пост-цонтент">
            ...
            <див класс="пост-фоотер">
                ...
                {%- иf цуррент_усер == пост.аутхор %} < a
                    хреf="{{ урл_фор ('.едит', ид=пост.ид) }}"
                        <спан класс="ла ел ла ел-примари">Измени</
                    спан> </ a > {%- елиf цуррент_усер.ис_администратор() %}
                    <а хреf="{{ урл_фор('.едит', ид=пост.ид) }}>

                <спан класс="ла ел ла ел-дангер">Измени [администратора]</
                спан> </ a > {%- ендиф %} </див> </ли> {%- ендфор %} </ул>
```

Ова промена додаје везу „Уреди“ свим постовима на логу чији је аутор тренутни корисник. За администраторе, линк се додаје свим о јавама. Администраторска веза је другачије стилизована као визуелни знак да је ово функција администрације. Слика 11-4 показује како везе за уређивање и сталну везу изгледају у ве претраживачу.



Слика 11-4. Уредите и сталне везе у посту на логу



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкуот 11x да исте проверили ову верзију апликације.

## ГЛАВА 12

### Фолловерс

Друштвено свесне ве апликације омогућавају корисницима да се повежу са другим корисницима. Различите апликације називају ове везе пратиоцима, пријатељима, контактима, везама или другарима, али функција је иста – ез о зира на име и у свим случајевима укључује праћење усмерених веза између парова корисника и коришћење ових веза у упити азе података.

У овом поглављу ћете научити како да примените функцију пратиоца за Фласки. Корисници ће моћи да „прате“ друге кориснике и иза јеру да филтрирају листу логова на почетној страници да и укључили само оне од корисника које прате.

#### Ревиситетед Релатионсхипс Релатионсхипс

Као што је о јашњено у [Поглављу 5](#), азе података успостављају везе између записа користећи релације. Однос један-према-више је најчешћи тип односа, где је запис повезан са листом повезаних записа. За имплементацију ове врсте односа, елементи на страни „много“ имају страни кључ који указује на повезани елемент на страни „један“. Пример апликације у свом тренутном стању укључује два односа један према више: један који повезује улоге корисника са листама корисника и други који повезује кориснике са постовима на јлогу чији су аутори.

Већина других типова односа може ити изведена из типа један-према-више. Однос „много-један“ је однос „један-на-више“ са становишта „много“ стране. Тип односа један-на-један је поједностављење односа један-према-више, где је страна „много“ ограничена да има највише један елемент. Једини тип односа који се не може имплементирати као једноставна варијација модела један-према-више је модел-више-према-више, који има листе елемената на о је стране. Овај однос је детаљно описан у следећем одељку.

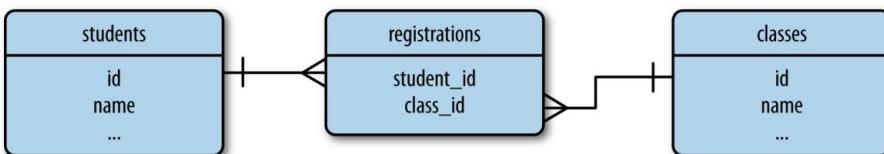
### Везе „много-према-више“ Релације

„један-према-више“, „много-према-један“ и „један-на-један“ имају арем једну страну са једним ентитетом, тако да се везе између повезаних записа имплементирају са страним кључевима који упућују на тај један елемент. Али како имплементирати однос у којем су оне стране „многа“ страна?

Размотрите класичан пример односа више-према-више: азу података о ученицима и часовима које похађају. Јасно је да не можете додати страни кључ разреду у тај ели ученика, јер ученик узима много часова — један страни кључ није довољан.

Исто тако, не можете додати страни кључ ученику у тај елу одељења, јер одељења имају више од једног ученика. Оне стране требају листу страних кључева.

Решење је додавање треће табеле у азу података, која се зове тајела асоцијације. Сада се однос „много-према-више“ може разложити на две релације „један-према-више“ из сваке од две оригиналне табеле до таје асоцијација. [Слика 12-1](#) показује како је представљен однос више-према-више између ученика и одељења.



Слика 12-1. Пример односа „много-према-више“.

Тајела асоцијација у овом примеру се зове регистрације. Сваки ред у овој табели представља индивидуалну регистрацију ученика у разреду.

Испитивање односа више-према-више је процес у два корака. Дајте исте до или листу часова које студент похађа, полазите од односа један-према-више између ученика и пријава и дојдите листу уписа за жељеног ученика. Затим се однос један-према-више између разреда и регистрација прелази у правцу више-према-један да и се до илје све класе повезане са регистрацијама које су преузете за ученика. Исто тако, дајте исте пронашли све ученике у одељењу, кренете од разреда и дојдите листу регистрација, а затим повежете ученике са тим регистрацијама.

Прелазак две релације за дојијање резултата упита звучи тешко, али за једноставну релацију као што је она у претходном примеру, СКЛАЛцхеми оавља већину послана. Следи код који представља однос много-према-више на [слици 12-1](#):

```
регистрације = д .Та ле('регистратионс',
    д .Цолумн('студент_ид', д .Интегер, д .ФореигнКеи('студентс.ид')),
    д .Цолумн('класс_ид', д .Интегер, д .ФореигнКеи('классес.ид'))
)
```

ученик разреда (д .Модел):

```
ид = д .Цолумн(д .Интегер, примарни_кеи =Труе)
наме = д .Цолумн(д .Стринг) классес =
д .релатионсхип('Клас', сецондари=регистратионс,
    ацкреф=д . ацкреф('студентс',
    лази= 'динамичан'), лази='динамичан')
```

**класс** Клас(д .Модел):

```
ид = д .Цолумн(д .Интегер, примарни_кључ =Труе)
наме = д .Цолумн(д .Стринг)
```

Однос је дефинисан истом конструкцијом д .релатионсхип() која се користи за релације један-према-више, али у случају односа више-према-више додатни секундарни аргумент мора ити постављен на тај елу асоцијација. Однос се може дефинисати у једном којој од ове две класе, при чему аргумент ацкреф води рачуна о излагању односа и са друге стране. Асоцијацијска тајела је дефинисана као једноставна тајела, а не као модел, пошто СКЛАЦХЕМИ интерно управља овом тајлом.

Однос класа користи семантику листе, што чини рад са односом много-према много конфигурисаним на овај начин изузетно лаким. С овим зиром на ученике са и класу ц, код који региструје ученика за разред је:

```
>>> с.классес.append(ц)
>>> д .сессион.адд(с)
```

Упити који наводе одељења за која је ученик с пријављен и листа ученика пријављених за одељење ц су такође веома једноставни:

```
>>> с.классес.all() >>>
ц.студентс.all()
```

Однос ученика доступан у моделу класе је онај који је дефинисан у аргументу д . ацкреф(). Имајте на уму да је у овом односу аргумент ацкреф проширен тако да има атрибут лази='динамиц', тако да овај е стране враћају упит који може да прихвати додатне филтере.

Ако ученик касније одлучи да одустане од класе ц, можете ажурирати азу података на следећи начин :

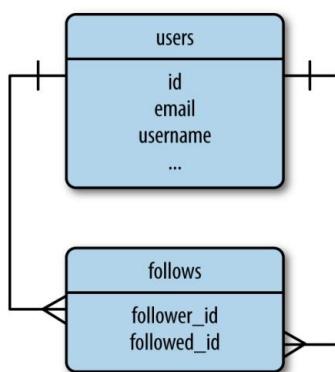
```
>>> с.классес.ремове(ц)
```

Референтни односи на се е. Однос

више-према-много може се користити за моделирање корисника који прате друге кориснике, али постоји про лем. У примеру ученика и одељења, постојала су два врло јасно дефинисана ентитета повезана та елом асоцијација. Међутим, за представљање корисника који прате друге кориснике, то су само корисници — не постоји други ентитет.

За однос у којем о е стране припадају истом столу се каже да је самореференцијалан. У овом случају, ентитети на левој страни односа су корисници, који се могу назвати „след еницима“. Ентитети са десне стране су такође корисници, али ово су „праћени“ корисници. Концептуално, самореферентни односи се не разликују од редовних веза, али о њима је теже размишљати.

[Слика 12-2](#) приказује дијаграм азе података за самореферентни однос који представља кориснике који прате друге корисника.



Слика 12-2. Пратиоци, однос много-према-многи

Та ела асоцијација у овом случају се зове како следи. Сваки ред у овој табели представља корисника који прати другог корисника. Однос један према више приказан на левој страни повезује кориснике са листом редова „прати“ у којима су они пратиоци. Однос један-према-више приказан на десној страни повезује кориснике са листом редова „прати“ у којима су корисници праћени.

Напредни односи „много-према-више“ Са

самореференцијалном релацијом „много-према-више“ конфигурисаном као што је приказано у претходном примеру, аза података може представљати след енике – али постоји једно ограничење. Уочијена потреба када се ради са односима много-према-многи је складиштење додатних података који се односе на везу између два ентитета. За однос пратилаца, може или корисно сачувати датум када је корисник почeo да прати другог корисника, јер ће то омогућити да се листе пратилаца представе хронолошким редоследом. Једино место ове ин

Та је асоцијација је интерна та је асоцијација којом у потпуности управља СКЛАЛцхеми.

Да исте могли да радите са прилагођеним подацима у односу, та је асоцијација мора ити унапређена у одговарајући модел коме апликација може да приступи. [Пример 12-1](#) приказује нову тајну асоцијација, представљену моделом Фолловс.

Пример 12-1. апп/моделс.пи: следећа тајна асоцијација као модел

[цласс](#) Фолловс(д .Модел):

```
_та_ленаме_ = 'фолловс'
фолловер_ид = д .Цолумн(д .Интигер, д .ФореигнКеи('усерс.ид'),
    примарни_кључ=Тачно)
фолловед_ид = д .Цолумн(д .Интигер, д .ФореигнКеи('усерс.ид'),
    примари_кеи=True)
временска_ознака = д .Цолумн(д .Датетиме, дефаулт=датетиме.утцнов)
```

СКЛАЛцхеми не може транспарентно да користи тајну асоцијација јер то неће дати апликацији приступ прилагођеним пољима у њој. Уместо тога, однос „много-према-више“ мора ити разложен на два основна односа један-према-више за леву и десну страну, а они морају ити дефинисани као стандардни односи. Ово је приказано у [примеру 12-2](#).

Пример 12-2. апп/моделс.пи: однос више-према-више имплементиран као два односа један-према-више

[цласс](#) Усер(УсерМикин, д .Модел):

```
...
# фолловед = д .релатионсхип('Фоллов',
    ектернал_кеис=[Фоллов.фолловер_ид],
    ацкреф=д . ацкреф('фолловер', лази='јоинед'),
    лази='динамиц', цасцаде=алл, делете-опрхан ) фолловерс
    = д .релатионсхип('Фоллов',
    ектернал_кеис=[Фоллов.фолловед_ид], ацкреф=д . ацкреф('фолловед', лази='јоинед'),
    лази='динамиц', цасцаде=алл, делете- сироче')
```

Овде се односи праћења и следећи дефинишу као појединачни односи један према више. Имајте на уму да је неопходно елиминисати ило какву двосмисленост између страних кључева тако што ћете у свакој релацији навести који страни кључ користити преко опционог аргумента страни\_кеис . Аргументи д . ацкреф() у овим релацијама не важе једни на друге; задње референце се примењују на модел Фолловс .

Лењи аргумент за задње референце је наведен као спојен. Овај лењи режим узрокује да се повезани о јекат одмах учита из упита за придрживање. На пример, ако корисник прати 100 других корисника, позивање усер.фолловед.алл() ће вратити листу од 100 инстанци праћења , где свака има својства след еника и праћене повратне референце постављене на одговарајуће кориснике. Режим лази=јоинед' омогућава да се све ово деси из једног упита азе података. Ако је лази постављена на подразумевану вредност селект, онда се пратилац и праћени корисници учитавају лењо када им се први пут приступи и сваки атри ут ће захтевати појединачни упит, што значи да и до ијање комплетне листе праћених корисника захтевало 100 додатних упита азе података .

Лењи аргумент на страни корисника о а односа има различите потре е. Они су на страни „једна“ и враћају страну „много“; овде се користи начин динамике , тако да атри ути односа враћају о јекте упита уместо да директно враћају ставке. Ово омогућава да се упиту додају додатни филтери пре него што се изврши.

Аргумент каскаде конфигурише како се радње извршene на родитељском о јекту шире на повезане о јекте. Пример каскадне опције је правило које каже да када се о јекат дода сесији азе података, сви о јекти повезани са њим кроз релације такође тре а аутоматски да се додају сесији. Подразумеване каскадне опције су прикладне за већину ситуација, али постоји један случај у коме подразумеване каскадне опције не функционишу до ро за овај однос „много према много“. Подразумевано каскадно понашање када се о јекат рише је постављање страног кључа у свим повезаним о јектима који се повезују са њим на нулту вредност. Али за та елу асоцијација, исправно понашање је рисање уноса који упућују на запис који је о рисан, јер то ефективно уништава везу. Ово ради опција каскаде рисања сирочета .



Вредност дата каскади је листа каскадних опција раздвојених зарезима. Ово је донекле з уњујуће, али опција под називом алл представља све каскадне опције осим делете-орпхан. Користећи вредност алл, делете-орпхан оставља укључене подразумеване каскадне опције и додаје понашање рисања за сирочад.

Апликација сада тре а да ради са две релације „један-према-много“ да и применила функцију „много-према-више“. Пошто су то операције које ће се морати често понављати, до ра је идеја креирати помоћне методе у моделу корисника за све могуће операције. Четири нове методе које контролишу овај однос приказане су у [Примеру 12-3](#).

Пример 12-3. апп/модел.пи: помоћне методе пратилаца

Корисник `класе` (д .Модел):

# ...

`деф фоллов(селф, усер):`

`ако није селф.ис_фолловинг(усер):`

`ф = Фоллов(фолловер=селф, фолловед =усер)`

`д .сессион.адд(ф)`

`деф унфоллов(селф, усер): ф`

`= селф.фолловед.фильтер_ и(фолловед_ид=усер.ид).фирст() иф ф:`

`д .сессион.делете(ф)`

`деф ис_фолловинг(селф, усер):`

`ако је усер.ид Ноне:`

`ретурн Фалсе`

`ретурн`

`селф.фолловед.фильтер_ и( фолловед_ид =усер.ид).фирст() није Ништа`

`деф ис_фолловед_ и(селф, усер):`

`ако је усер.ид Ноне: ретурн`

`Фалсе ретурн`

`селф.фолловер.фильтер_ и( фолловер_ид`

`=усер.ид).фирст() није Ништа`

Метод `фоллов()` ручно у ацује инстанцу Прати у та елу асоцијација која повезује пратиоца са праћеним корисником, дајући апликацији прилику да постави прилагођено поље. Два корисника која се повезују се ручно додељују новој инстанци Фоллов у њеном конструктору, а затим се о јекат додаје у сесију азе података као и о ично. Имајте на уму да нема потребе да ручно постављате поље временске ознаке јер је дефинисано са подразумеваном вредношћу која поставља тренутни датум и време. Тхе

`унфоллов()` метода користи однос праћења да лоцира инстанцу Прати која повезује корисника са праћеним корисником који трећа да уде искључен. Да и се унишитила веза између два корисника, о јекат Прати се једноставно реше. Методе `ис_фолловинг()` и `ис_фолловед_ и()` претражују леву и десну релацију један према више, респективно, за датог корисника и враћају Тачно ако је корисник пронађен. О а о ез еђују да је датом кориснику додељен ИД пре издавања упита, да и се из егле грешке ако је наведен корисник који је креiran, али још увек није уписан у азу података.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит чхекаут 12а да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код.

Део азе података ове функције је сада завршен. Можете пронаћи јединични тест који веж а нови однос азе података у спремишту извornог кода на ГитХу -у.

## Пратиоци на страницама Про ле

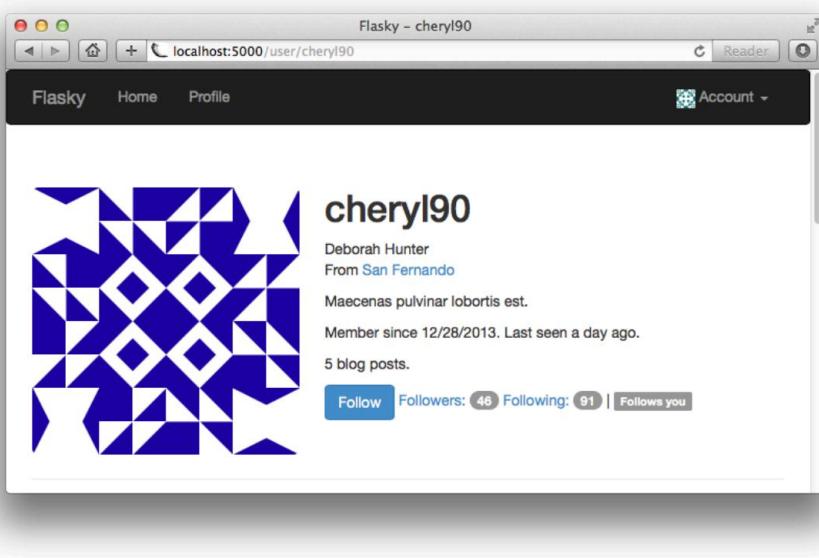
Страница профила корисника тре а да има дугме „Прати“ ако корисник који га гледа није пратилац или дугме „Прекини праћење“ ако је корисник пратилац. Такође је леп додатак да се прикаже број пратилаца и праћених, да се прикаже листа пратилаца и праћених корисника и да се покаже знак „Прати вас“ када је то прикладно. Промене ша лона корисничког профила приказане су у [примеру 12-4. Слика 12-3](#) приказује како додаци изгледају на страницама профила.

Пример 12-4. ап/темплатес/усер.хтмл: по ољшања пратиоца у заглављу профила корисника

```
{% if цуррент_усер.цан(Пермиссион.ФОЛЛОВ) и усер != цуррент_усер %}
    {% if нот цуррент_усер.ис_фолловинг(усер) %} < a
        хреф="{{ урл_фор ('.фоллов', усернаме=усер.усернаме ) }}" класс=" тн  тн-
            примари "Прати</а> { % остало %} <a хреф="{{ урл_фор('унфоллов',
            корисничко име=усер.усернаме ) }}">
                класс=" тн  тн-дефайл">Опозови праћење</а>
            {% ендиф %}
        {% ендиф %} <
    а хреф="{{ урл_фор ('.фолловерс', корисничко име=усер.усернаме ) }}>
        Пратиоци: <спан класс=" адге">{{ усер.фолловерс.коунт() }}</спан> </а> <a
        хреф="{{ урл_фор('.фолловед_ и', усернаме=усер.усернаме ) }}>
```

Следеће: <спан класс=" адге">{{ усер.фолловед.коунт() }}</спан> </а> {% иф
 цуррент\_усер.ис\_аутхентицатед и усер != цуррент\_усер и

 усер.ис\_фолловинг(цуррент\_усер) %}
| <спан класс="ла ел ла ел-дефайл">Прати вас</спан> {% ендиф %}



Слика 12-3. Пратиоци на про ле страници

Постоје четири нове крајње тачке дефинисане у овим променама ша лона. Рута /фоллов/ <усер-наме> се позива када корисник кликне на дугме „Фоллов“ на страници профила другог корисника. Имплементација је приказана у [Примеру 12-5](#).

Пример 12-5. апп/маин/виевс.пи: прати руту и функцију прегледа

```
@маин.роуте('/фоллов/<корисничко
име>') @логин_реакуиред @
пермиссион_реакуиред(Пермиссион.ФОЛЛОВ)
деф фоллов(усернаме): усер =
    Усер.куери.филтер_ и(усернаме=усернаме).фирст() ако је
    корисник Ништа: фласх('Неважећи корисник.') ретурн
        редирект(урл_фор('.индек'))
    иф цуррент_усер.ис_фолловинг(усер):
        фласх(' Већ пратите овог корисника.') ретурн
        редирект(урл_фор('.усер', усернаме=усернаме))
    цуррент_усер.фоллов(усер)
    д .сесион.цоммит() фласх(
        Сада пратите %c.% корисничко име) ретурн
    редиреクト(урл_фор('.усер', усернаме=усернаме))
```

Ова функција приказа учитава захтеваног корисника, проверава да ли је важећи и да га већ не прати пријављени корисник, а затим позива помоћну функцију фоллов() у

модел корисника за успостављање везе. Рута /унфоллов/**<усернаме>** је имплементирана на сличан начин.

Рута /фолловерс/**<усернаме>** се позива када корисник кликне на **рој** пратилаца другог корисника на страници профила. Имплементација је приказана у [Примеру 12-6](#).

Пример 12-6. апп/маин/виевс.пи: рута пратилаца и функција прегледа

```
@маин.роуте('/фолловерс/<корисничко
име>') деф фолловерс( усернаме): усер =
    Усер.куери.филтер_ (усернаме=усернаме).фирст() ако је
    корисник Ништа: фласх('Неважећи корисник.')
    редирект(урл_фор(.индек')) паге = рекуест.аргс.гет('паге',
    1, типе=инт) пагинатион = усер.фолловерс.пагинате(
        страница, пер_паге=цуррент.апп.ценфиг['ФЛАСКИ_ФОЛЛОВЕРС_ПЕР_ПАГЕ'],
        еррор_аут=Фалсе) фолловс = [ {'усер': item.фолловер, 'тиместамп':
        item.тиместамп} за ставку у пагинатион.итемс] ретурн
        рендер_темплате('фолловерс.хтмл', усер=усер, титле="Фолловерс
        оф",
        ендпоинт='фолловерс', пагинатион=пагинатион,
        фолловс=фолловс )
```

Ова функција учитава и потврђује траженог корисника, затим пагинира његов однос пратилаца користећи исте технике научене у [поглављу 11](#). Пошто упит за пратиоце враћа инстанце Прати, листа се конвертује у другу листу која има поља за корисника и временску ознаку у сваком уносу тако да рендеровање је једноставније.

Ша лон који приказује листу пратилаца може ити написан генерички тако да се може користити за листе пратилаца и праћених корисника. Ша лон прима корисника, наслов странице, крајњу тачку за коришћење у везама за пагинацију, ојекат пагинације и листу резултата.

Крајња тачка фолловед и је скоро идентична. Једина разлика је у томе што се листа корисника дојија из односа усер.фолловед. Аргументи ша лона су takoђе прилагођени у складу са тим.

Ша лон фолловерс.хтмл је имплементиран са тајелом са две колоне која приказује корисничка имена и њихове аватаре на левој страни и Фласк-Момент временске ознаке на десној страни. Можете консултовати спремиште изворног кода на ГитХуу да исти детаљно проучили имплементацију.



Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит цхеџкоут 12 да исти проверили ову верзију апликације.

## Упитивање праћених постова помоћу придрживања ази података

Почетна страница апликације тренутно приказује све постове у ази података у опадајућем хронолошком редоследу. Пошто је функција пратилаца сада завршена, ило и лепо да се корисницима да могућност да виде постове на логу само од корисника које прате.

Очигледан начин да учитате све постове чији су аутори праћени корисници је да прво до ијете листу тих корисника, а затим до ијете постове од сваког и сортирате их у једну листу. Наравно, тај приступ није до бо размјеран; напор потреан за до ијање ове ком иноване листе ће расти како аза података уде расла, а операције као што је пагинација не могу се ефикасно обављати. Овај пролем је опште познат као „Н+1 пролем“, јер рад са азом података на овај начин захтева издавање Н+1 упита азе података, при чему је Н број резултата које је вратио први упит. Кључ за до ијање постова на логу са до рим перформансама вез о зира на величину азе података је да све то урадите једним упитом.

Операција азе података која то може да уради назива се спајање. Операција спајања узима две или више табела и проналази све ком инације редова које задовољавају дати услов. До ијени ком иновани редови се у ацују у привремену табелу која је резултат спајања. Најолнији начин да о јасните како функционишу спојеви је кроз пример.

**Тајела 12-1** приказује пример табеле корисника са три корисника.

Тајела 12-1. табела корисника

ид корисничко име

- 1 Јохн
- 2 Сузан
- 3 Давид

**Тајела 12-2** приказује одговарајућу табелу постова, са неким постовима на логу.

Тајела 12-2. постова табела

ид аутор\_ид тело

- |   |   |                         |
|---|---|-------------------------|
| 1 | 2 | Пост на логу Сузан      |
| 2 | 1 | Ојава на логу Јохна     |
| 3 | 3 | Блог пост од Давида     |
| 4 | 1 | Други лог пост од Јохна |

Конечно, **тајела 12-3** показује ко кога прати. У овој табели можете видети да Џон прати Давида, Сузан прати Џона и Дејвида, а Давид не прати никога.

Та ела 12-3. прати та елу

фолловер\_ид фолловед\_ид

1	3
2	1
2	3

Да исте до или листу постова од стране корисника праћених корисником сусан, постови и пратиоци та еле морају ити ком иноване. Прво се филтрира следећа та ела да и се задржали само редови који имају Сусан као пратиоца, што су у овом примеру последња два реда. Затим темпорари спојна та ела је креирана од свих могућих ком иниција редова из постова и филтрирано прати та еле у којима је аутор\_ид поста исти као фолловед\_ид од следећег, ефективно ирађују све постове који се појављују на листи корисници Сусан прати. Та ела 12-4 приказује резултат операције спајања. Колумни који су коришћени за извођење спајања означени су са знаком \* у овој табели.

Та ела 12-4. Спојени сто

ид аутор_ид* тело	фолловер_ид фолловед_ид*
2 1 О јава на логу Јохна	2 1
3 3 Блог пост од Давида	2 3
4 1 Други лог пост од Јована 2	1

Ова та ела садржи управо листу лог постова чији су аутори корисници које Сузан прати-инг. Флакс-СКЛАлцхеми упит који изводи операцију спајања како је описано је прилично сложено:

```
ретурн д .сессион.куери(Пост).селект_фром(Фоллов).\n    филтер_ и(фолловер_ид=селф.ид).\n    јоин(Пост, Фоллов.фолловед_ид == Пост.аутхор_ид)
```

Сви упити које сте до сада видели почињу од атријута упита модела то се испитује. Тада формат не функционише до по за овај упит, јер упит треба да врати редове постова, али прва операција коју треба урадити је да се примени а филтрирајте на следећу та елу. Дакле, уместо тога се користи основнији облик упита. У потпуности разумете овај упит, сваки део треба посматрати појединачно:

- д .сессион.куери(Пост) наводи да ће ово ити упит који враћа О јавите ојекте.
- селект\_фром(Фоллов) каже да упит починje моделом Фоллов.
- филтер\_ и(фолловер\_ид=селф.ид) врши филтрирање следеће та еле према корисник пратиоца.

- јоин(Пост, Фоллов.фолловед\_ид == Пост.аутхор\_ид) спаја резултате филтер\_ и() са објектима Пост .

Упит се може поједноставити заменом редоследа филтера и спајања:

```
ретурн Пост.куери.јоин(Фоллов, Фоллов.фолловед_ид == Пост.аутхор_ид)\n    .филтер(Фоллов.фолловер_ид ==セルフ.ид)
```

Прво издавање операције придрживања значи да се упит може покренути из Пост.куери-а, тако да су сада једина два филтера која треба да се примене јоин() и филтер(). Може изгледати да и право спајање, а затим филтрирање ило више посла, али у стварности су ова два упита еквивалентна. СКЛАЛЦХЕМИ прво прикупља све филтере, а затим генерише упит на најефикаснији начин. Изворне СКЛ инструкције за ова два упита су скоро идентичне, нешто што можете потврдити штампањем ојекта упита конвертованог у стринг (тј. принт(стр(куери))). Коначна верзија овог упита се додаје моделу Пост , као што је приказано у [Примеру 12-7](#).

Пример 12-7. апп/моделс.пи: до ијање праћених постова

```
класа Корисник (д .Модел):\n# ...\n@пропрети\nдеф фолловед_постс (セルф):\n    ретурн Пост.куери.јоин(Фоллов, Фоллов.фолловед_ид == Пост.аутхор_ид)\n        .филтер(Фоллов.фолловер_ид ==セルフ.ид)
```

Имајте на уму да је метода фолловед\_постс() дефинисана као својство тако да јој није потребан(). На тај начин сви односи имају конзистентну синтаксу.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 12ц да исте проверили ову верзију апликације.

Спојеви су изузетно тешки за омотавање главе; можда ћете морати да експериментишишете са примером кода у љусци пре него што све то потоне.

## Приказ праћених постова на почетној страници

Почетна страница сада може дати корисницима избор да погледају све постове на логу или само оне које прате корисници. [Пример 12-8](#) показује како се овај избор спроводи.

Пример 12-8. апп/маин/виевс.пи: приказује све постове или постове које пратите

```
@маин.роуте('/' метходс = ['ГЕТ', 'ПОСТ']) деф индек():

...
# схов_фолловед = Фалсе
ако је цуррент_усер.ис_аутхентицитет:
    схов_фолловед = оол(рекуест.циокиес.гет('схов_фолловед', '')) иф
    схов_фолловед: куери = цуррент_усер.фолловед_постс елсе:

    куери = Пост.куери
    пагинација = куери.ордер_ (Пост.тиместамп.десц()).пагинате(
        страница, пер_паге=циуррент_апп.ционфиг['ФЛАСКИ_ПОСТС_ПЕР_ПАГЕ'],
        еррор_аут=Фалсе) постови = пагинатион.итемс ретурн
        рендер_темплате('индек.хтмл', форм=форм, постс=постс,
        схов_фолловед=схов_фолловед, пагинатион=пагинатион)
```

Из овог приказивања свих или праћених постова се чува у колачићу који се зове схов\_фолловед који, када је постављен на непразан стринг, означава да трећи да се приказују само праћени постови. Колачић се чувају у оквиру захтева као речник рекуест.циокиес. Вредност низа колачића се конвертује у Боолеан, а на основу њене вредности локална променљива упита се поставља на упит који доји комплетну или филтрирану листу постова на логу. Да се приказали сви постови, користи се упит највишег нивоа Пост.куери, а недавно додато својство Усер.фолловед\_постс се користи када листа трећи да буде ограничена на праћене кориснике. Упит сачуван у локалној променљивој упита се затим пагинира и резултати се шаљу у шаблон као и раније.

Колачић схов\_фолловед је постављен на две нове руте, приказане у Примеру 12-9.

Пример 12-9. апп/маин/виевс.пи: из овог свих или праћених постова

```
@маин.роуте('/алл')
@логин_рекуире деф
схов_алл(): респ =
    маке_респонсе(редирект(урл_фор('.индек')))
    респ.сет_циокие('схов_фолловед', '', мак_аге=30*24*60 *60) # 30 дана враћања одн

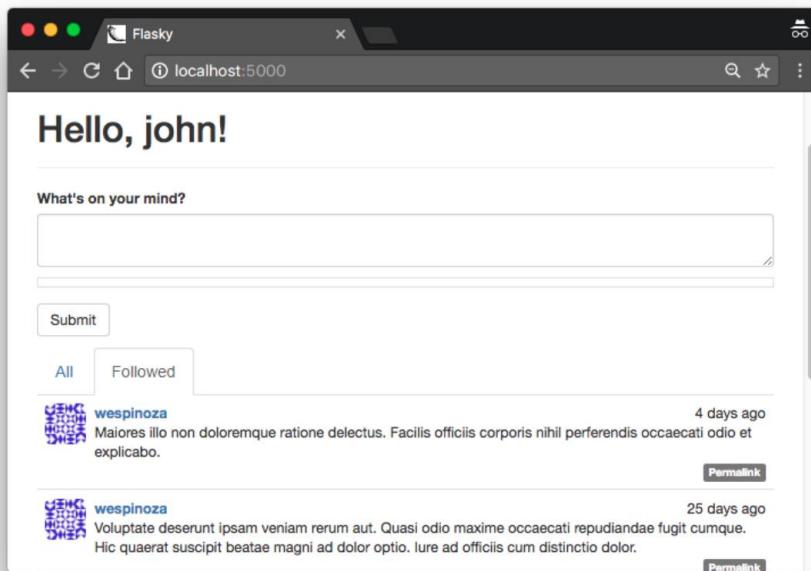
@маин.роуте('/фолловед')
@логин_рекуире деф
схов_фолловед(): респ =
    маке_респонсе(редирект(урл_фор('.индек')))
    респ.сет_циокие('схов_фолловед', '1', мак_аге=30*24*60*60) # 30 дана враћања одн
```

Везе ка овим путама се додају у ша лон почетне странице. Када се позову, колачић сков\_фолловед се поставља на одговарајућу вредност и издаје се преусмеравање назад на почетну страницу.

Колачићи се могу подесити само на објекту одговора, тако да ове руте морају да креирају објекат одговора преко `makes_response()` уместо да допусте Фласку да то уради.

Функција `set_cookie()` узима име колачића и вредност као прва два аргумента. Опционални аргумент `max_age` поставља број секунди до истека колачића. Ако се овај аргумент не укључи, колачић ће истећи када се затвори прозор претраживача. У овом случају се поставља максимална старост од 30 дана тако да се поставка памти чак и ако се корисник не врати у апликацију неколико дана.

Промене у ша лону додају две навигационе картице на врху странице које позивају `/all` или `/followers` руте да су се поставила исправна подешавања у сесији. Можете детаљно прегледати промене ша лона у спремишту извornог кода на ГитХуу. [Слика 12-4](#) показује како почетна страница изгледа са овим променама.



Слика 12-4. Прати постове на почетној страници



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 12д да исте проверили ову верзију апликације.

Ако испротестате апликацију у овом тренутку и пређете на следећи листу постова, приметићете да се ваши постови не појављују на листи. Ово је наравно тачно, јер корисници нису сами се и следе еници.

Иако упити функционишу како је дизајнирано, већина корисника ће очекивати да види своје постове када гледају ојаве својих пријатеља. Најлакши начин да решите овај проблем је да региструјете све кориснике као сопствене пратиоце у тренутку када су креирани.

Овај трик је приказан у [примеру 12-10](#).

Пример 12-10. апп/модел.пи: прављење корисника сопственим пратиоцима када су креирани

```
класс Усер(УсерМикин, д .Модел):
    # ...
    деф __инит__(селф, **кваргс):
        # ...
        селф.фоллов(селф)
```

Нажалост, вероватно имате неколико корисника у ази података који су већ креирани и не прате сами се је. Ако је аза података мала и лако се регенерише, онда се може изјаснити и поново креирати, али ако то није опција, онда је додавање функције ажурирања која поправља постојеће кориснике право решење. Ово је приказано у [примеру 12-11](#).

Пример 12-11. апп/модел.пи: прављење корисника сопственим следећима

```
класс Усер(УсерМикин, д .Модел):
    # ...
    @статичметод
    деф add_селф_фолловс():
        за корисника у Усер.куери.алл():
            ако није усер.ис_фолловинг(усер):
                усер.фоллов(усер)
            д .сесион.адд(усер)
            д .сесион.коммит()
        # ...
```

Сада се аза података може ажурирати покретањем претходне функције примера из љуске:

```
(венв) $ флашк скелл
>>> Усер.add_селф_фолловс()
```

Креирање функција које уводе ажурирања у азу података је уочено ичайена техника која се користи за ажурирање апликација које су распоређене, јер је покретање ажурирања са скриптом мање подложно грешкама него ручно ажурирање аза података. У [поглављу 17](#) видећете како се ова функција и друге сличне њој могу уградити у скрипту за примену.

Омогућавање да сви корисници сами прате апликацију чини апликацију употребе љивијом, али ова промена доноси неколико компликација. Број пратилаца и праћених корисника приказан на страници корисничког профиле сада је повећан за један због веза за самопратење. Бројеве треће смањити за један да ће или тачни, што је лако урадити директно у шаблону приказивањем {{ user.followers.count() - 1 }} и {{ user.followed.count() - 1 }}. Листе пратилаца и праћених корисника такође морају да прилагођене тако да не приказују истог корисника, што је још један једноставан задатак који треба да урадити у шаблону са условом. Коначно, на све јединице тестове који проверавају број пратилаца такође утичу везе самопратећих и морају се прилагодити тако да узимају у обзир самослед јенике.



Ако сте клонирали Гит спремиште апликације на ГитХубу, можете покренути гит цхецкоут 12е да бисте проверили ову верзију апликације.

У следећем поглављу ће имплементиран подсистем корисничких коментара — још једна веома важна карактеристика друштвено свесних апликација.

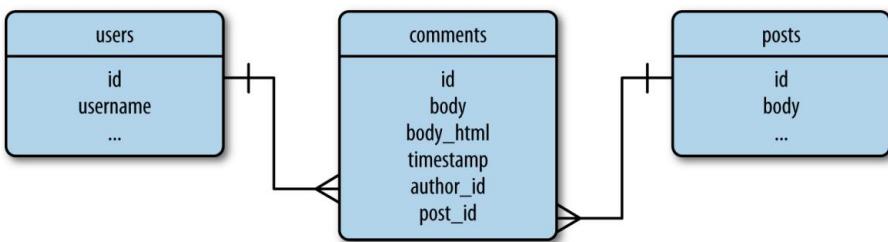
## ГЛАВА 13

### Коментари корисника

Омогућавање корисницима интеракције кључно је за успех платформе за друштвено логовање. У овом поглављу ћете научити како да имплементирате коментаре корисника. Представљене технике су доволно генеричке да се могу директно применити на великији друштвено омогућених апликација.

#### База података Репрезентација коментара

Коментари се не разликују много од постова на логу. Оне имају тело, аутора и временску ознаку, а у овој конкретној имплементацији оне су написана Маркдован синтаксом. Слика 13-1 приказује дијаграм табела коментара и њених односа са другим табелама у бази података.



Слика 13-1. Репрезентација коментара на логу у бази података

Коментари се примењују на одређене постове на логу, тако да је дефинисан однос један-према-више из табеле постова. Овај однос се може користити за доирање листе коментара повезаних са одређеним постом на логу.

Табела коментара је такође у односу један-према-више са табелом корисника. Овај однос даје приступ свим коментарима које је направио корисник, и посредно како

много коментара које је корисник написао, информација која може ити занимљива за приказ на страницама корисничког профила. Дефиниција модела коментара је приказана у [примеру 13-1](#).

Пример 13-1. апп/моделс.пи: модел коментара

```
цлас Цоммент(д .Модел):
    _та_ленаме_ = 'комментс'
    ид = д .Цолумн(д .Интегер, примари_кеи =Труе)
    оди = д .Цолумн(д .Текст) оди_хтмл =
        д .Цолумн(д .Текст) тиместамп =
            д .Цолумн(д .ДатеТиме, индекс =Труе, дефаулт=датетиме.утцнов) диса лед =
                д .Цолумн(д .Боолеан) аутхор_ид = д .Цолумн(д .Интегер,
                    д .ФореигнКеи('усерс.ид')) пост_ид = д .Цолумн(д .Интегер,
                        д .ФореигнКеи('постс.ид'))

    @статицметод
    деф он_цхангед_ оди(циль, вредност, стара вредност, покретач):
        дозвољено_tags = ['а', 'а р', 'акроним', ' ', 'код', 'ем', 'и', 'јако']
        таргет.оди_хтмл =
            леацх.линкифи(леацх.цлеан(маркдовн(валуе, оутпут_формат='хтмл'),
                tags=алловед_tags, стрип=Труе))

    д .евент.лиsten(Цоммент.оди, 'сет', Цоммент.он_цхангед_ оди)
```

Атри ути модела коментара су скоро исти као и атри ути Поста. Један додатак је диса лед поље, Боолеан који ће користити модератори да потисну коментаре који су неприкладни или увредљиви. Попут постова на логу, коментари дефинишу догађај који се покреће сваки пут када се поље тела промени, аутоматизујући приказивање Маркдовн текста у ХТМЛ. Процес је идентичан ономе што је урађено за постове на логу у [Поглављу 11](#), или пошто су коментари о ично кратки, листа ХТМЛ ознака које су дозвољене у конверзији из Маркдовн-а је рестриктивнија, ознаке које се односе на параграф су уклоњене и остају само ознаке за форматирање знакова.

Да и се довршиле промене азе података, модели Усер и Пост морају дефинисати односе један према више са та елом коментара, као што је приказано у [Примеру 13-2](#).

Пример 13-2. апп/моделс.пи: односи један према више корисника и постови за коментари

```
Корисник класе(д .Модел):
    #
    ...

    комментс = д .релационсхип('Цоммент', ацкреф='аутхор', лази='динамиц')

цлас Пост(д .Модел): #
    ...
    комментс = д .релационсхип('Цоммент', ацкреф='пост', лази='динамиц')
```

## Подношење и приказ коментара

У овој апликацији, коментари се приказују на појединачним страницама логова које су додате као трајне везе у [поглављу 11](#). О разац за подношење је такође укључен на овим страницама.

[Пример 13-3](#) показује вео разац који ће се користити за унос коментара — изузетно једноставан о разац који има само текстуално поље и дугме за слање.

Пример 13-3. апп/майн/формс.пи: о разац за унос коментара

```
класс ЦомментФорм(ФлакСформ):
    оди = СтингФиелд(), валидаторс=[ДатаРекуиред()])
    су мит = Су митФиелд('Су мит')
```

[Пример 13-4](#) приказује ажурирану руту /пост/<инт:ид> са подршком за коментаре.

Пример 13-4. апп/майн/виеовс.пи: подршка за коментаре на логу

```
@майн.роуте('/пост/<инт:ид>', методс=['ГЕТ', 'ПОСТ']) деф
пост(ид): пост = Пост.куери.гет_оп_404(ид) форм = ЦомментФорм()
    ако форм.валидате_он_су мит(): цоммент =
        Цоммент( оди=форм. оди.дата,
                    пост=пост,
                    аутхор=цуррент_усер._гет_цуррент_о_јеџt())
        д .сессион.адд(цоммент) д .сессион.цоммит() фласх('Ваш коментар
            је о явљен.') ретурн редиреџт(урл_фор(.пост', ид=) пост.ид, паге=-1)
        паге = рекуест.аргс.гет('паге', 1, типе=инт) иф паге == -1: паге =
            (пост.цомментс.цоунт() - 1) //\
        цуррент_апп.ценфиг['ФЛАСКИ_ЦОММЕНТС_ПЕР_ПАГЕ'] + 1 пагинација =
        пост.цомментс.ордер_ и (Цоммент.тиместамп.асц()).пагинате( паге,
            пер_паге=цуррент_апп.ценфиг['ФЛАСКИ_ЦОММЕНТС_ПЕР_ПАГЕ'],
            еррор_аутс= Фалсе страница .итемс враћају
            рендер_темплате('пост.хтмл', постс=[пост], форм=форм,
                            коментари=коментари, пагинација=пагинација)
```

Ова функција приказа инстанцира о разац за коментаре и шаље га на пост.хтмл ша лон за приказивање. Логика која у ацује нови коментар када се о разац пошаље је слична руковању лог постовима. Као иу случају Пост , аутор коментара не може ити постављен директно на цуррент\_усер јер је ово проки о јекат контекстуалне променљиве. Израз цуррент\_усер.\_гет\_цуррент\_о\_јеџt() враћа стварни о јекат корисника .

Коментари су сортирани по временској ознаки хронолошким редоследом, тако да се нови коментари увек додају на дно листе. Када се унесе нови коментар, преусмеравање које завршава захтев се враћа на исту URL адресу, али функција url\_for() поставља страницу на -1, посебно јер странице који се користи за захтевање последње странице коментара тако да управо унети коментар се види на страници. Када се рој странице дојде из низа упита и утврди да је -1, врши се прорачун са ројем коментара и величином странице да и се дојде ио стварни рој странице

Листа коментара повезаних са ојавом дојда се путем односа један-према-више пост.комментс, сортирана по временској ознаки коментара и пагинирана истим техникама које се користе за постове налогу. Коментари и ојекат пагинације се шаљу у шалон за приказивање. Променљива конфигурације ФЛАСКИ\_ЦОММЕНТС\_ПЕР\_ПАГЕ је додата у цон г.пи да контролише величину сваке странице коментара.

Рендеровање коментара је дефинисано у новом шалону, \_комментс.хтмл, који је сличан \_постс.хтмл, али користи другачији скуп ЦСС класа. Овај шалон је укључен у \_постс.хтмл испод тела поста, након чега следи позив макроа за пагинацију.

Можете прегледати промене шалона у ГитХу спремишту апликације.

Дајте доверили ову функцију, постови нају приказани на почетној страници и страницама профила трејају везе до страница са коментарима. Ово је приказано у примеру 13-5.

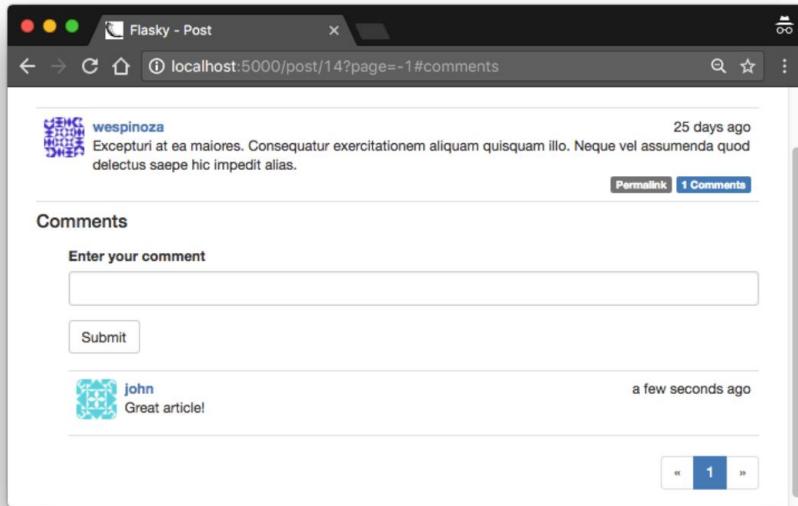
Пример 13-5. \_апп/темплатес/\_постс.хтмл: веза до коментара нају лог постовима

```
< a href="{{ url_for('.post', id=post.id ) }}#commentc "> <спан
    class="ла ел ла ел-примари">
        {{ post.commentc.цоунт() }} Коментари </
    спан> </ a>
```

Ојратите пажњу на то како текст везе укључује рој коментара, који се лако дојда из односа један-према-више између постова и таја ела са коментарима користећи СКЛАДЦЕМИ филтер цоунт().

Такође је занимљива структура везе до странице са коментарима, која је направљена као трајна веза за пост са додатком #комментс суфиксом. Овај последњи део назива се фрагмент УРЛ-а и користи се за означавање почетне позиције померања странице. Већ претраживач тражи елемент са датим ИД-ом и померају страницу тако да се тај елемент појави на врху странице. Ова почетна позиција је постављена на наслов „Коментари“ у шалону пост.хтмл, који је написан као <x4 id="commentc">Коментари</x4>.

Слика 13-2 показује како се коментари појављују на страници.



Слика 13-2. Коментари на логове

Додатна промена је направљена у макро пагинацији. Везама за пагинацију за коментаре такође треба додат #комментс фрагмент, тако да је аргумент фрагмената додат макроу и прослеђен у позивању макроа из шаблона пост.хтмл.



Ако сте клонирали Гит спремиште апликације на ГитХубу, можете покренути гит цхеџкуот 1за да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију базе података, па не заборавите да покренете надоградњу флакс док након што проверите код.

## Цоммент Модератион

У [поглављу 9](#) дефинисана је листа корисничких улога, свака са листом дозвола. Једна од дозвола је имала Пермиссион.МОДЕРАТЕ, која корисницима који га имају у својим улогама даје моћ да модерирају коментаре других.

Ова функција ће бити изложена као веза на траци за навигацију која се појављује само корисницима којима је дозвољено да је користе. Ово се ради у шаблону ace.хтмл користећи услов, као што је приказано у [Примеру 13-6](#).

Пример 13-6. апп/тэмплатес/ асе.хтмл: Веза за модерирање коментара на траци за навигацију

```
...
{%
    иф цуррент_усер.цан(Пермиссион.МОДЕРАТЕ) %}
<ли>< а href="{{ урл_фор ('маин.модерате') }}> Модерирај коментаре</а></ли> {%
    ендиф %}
...
}
```

Страница за модерирање приказује коментаре за све постове на истој листи, при чему су први приказани најновији коментари. Испод сваког коментара налази се дугме које може да пређацује атријут онемогућено . /модерате ruta је приказана у [Примеру 13-7.](#)

Пример 13-7. апп/маин/виеvс.пи: путања за модерирање коментара

```
@маин.роуте('/модерате')
@логин_рекуиред
@пермиссион_рекуиред ( Пермиссион.МОДЕРАТЕ)
деф модерате(): паге = рекуест.аргс.гет('паге', 1,
типе=инт) пагинатион =
Цоммент.куери.ордер_ и( Цоммент.тиместамп.десц()).пагинате( паге,
пер_паге=цуррент_апп.ценфиг['ФЛАСКИ_ЦОММЕНТС_ПЕР_ПАГЕ'],
еррор_оут=Фалс) цомментс = пагинатион.итемс ретурн
рендер_тэмплате('модерате.хтмл', цомментс=комментс,
пагинатион=пагинатион, паге=паге)
```

Ово је веома једноставна функција која чита страницу коментара из базе података и прослеђује их шаљену за рендеровање. Заједно са коментарима, шаљен прими ојекат пагинације и тренутнијије странице.

Шаљен модерате.хтмл, приказан у [Примеру 13-8](#), такође је једноставан јер се ослања на подешаљен \_комментс.хтмл креiran раније за приказивање коментаре.

Пример 13-8. апп/тэмплатес/модерате.хтмл: шаљен за модерирање коментара

```
{% проширује " асе.хтмл"
%} {% импорт "_мацрос.хтмл" као макроа %}

{% лоцк титле %}Фласки – Модерирање коментара{% енд лоцк %}

{% лоцк паге_центент %}
<див класс="паге-хеадер">
    <x1>Модерирање коментара</
    <x1> </див> {% сет модерате = Тачно
%} {% инклуде '_комментс.хтмл' %}
{% иф пагинација %} <див
класс="пагинатион">
```

```
 {{ мацрос.пагинатион_видгет(пагинатион, '.модерате') }} </див>
{%
    ендиф %
} {% енд  лоцк %}
```

Овај ша лон одлаже приказивање коментара на ша лон \_цомментс.хтмл, али пре него што преда контролу подређеном ша лону користи Јиња2-ову сет директиву да дефинише умерену променљиву ша лона постављену на Тачно. Ову променљиву користи ша лон \_цом-ментс.хтмл да и се утврдило да ли је потребно приказати карактеристике модерације.

Део ша лона \_цомментс.хтмл који приказује тело сваког коментара треба да се измени на два начина. За редовне кориснике (када умерена променљива није подешена), сви коментари који су означени као онемогућени треба да буду потиснути. За модераторе (када је модерате постављено на Тачно), тело коментара мора ити приказано без озира на стање онемогућено, а испод тела треба да буде укључено дугме за промену стања. **Пример 13-9** показује ове промене.

Пример 13-9. апп/темплатес/\_цомментс.хтмл: приказивање тела коментара

```
...
<див класс="цоммент- оди">
    {% иф цоммент.диса лед %}
    <п></п><и>Овај коментар је онемогућио модератор.</и></п> {% ендиф %}
    {% иф модерате оп нот цоммент.диса лед %} {% иф цоммент. оди_хтмл
%} {{ цоммент. оди_хтмл | сигурно }} {% остало %} {{ цоммент. оди }} {% ендиф %}

    {% ендиф %}
</див> {% иф
модерате %} < p>
    {% иф
        цоммент.диса лед %} <a
            класс=" тн тн-дефаулт тн-кс" хреф="{{ урл_фор('.модерате_ена ле'),
                ид=цоммент.ид, паге=паге })}">Оногући </a> {% еlse %} <а класс=" тн тн-
        дангер тн-кс" хреф="{{ урл_фор('.модерате_диса ле', ид=цоммент.ид,
                паге=паге })}">Онемогући</a>

    {% ендиф %}
    {% ендиф %}
...
```

Са овим променама, корисници ће видети кратко озвештење за онемогућене коментаре. Модератори ће видети и озвештење и тело коментара. Модератори ће такође видети дугме за преавивање стања онемогућено испод сваког коментара. Дугме позива једно од два нова

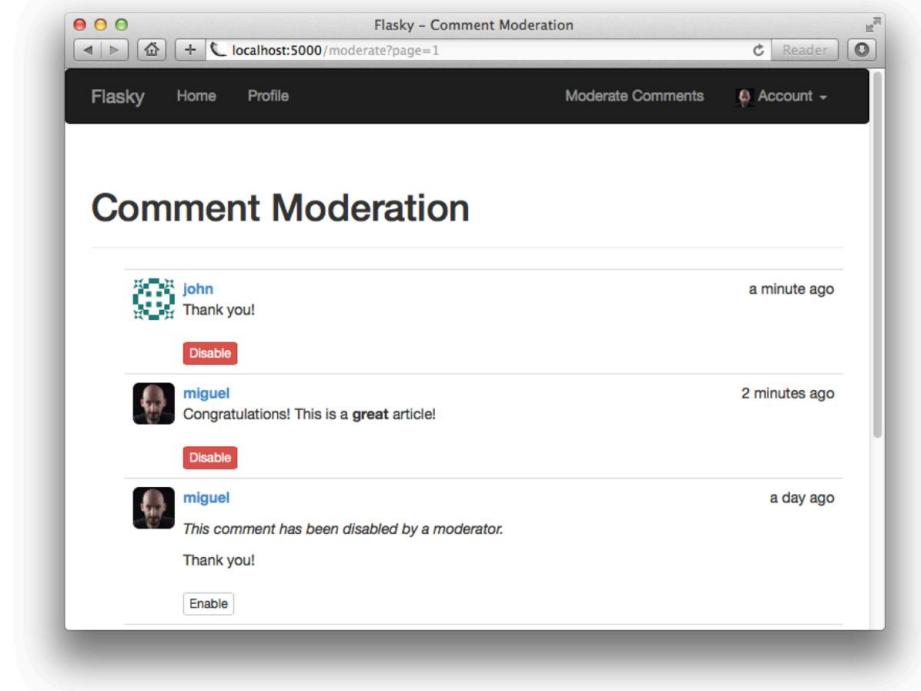
руте, у зависности од тога у које од два могућа стања се коментар мења.  
**Пример 13-10** показује како су ове руте дефинисане.

Пример 13-10. апп/маин/виевс.пи: руте за модерирање коментара

```
@маин.роуте('/модерате/ена ле/<инт:ид>')
@логин_рекуиред @
пермисион_рекуиред(Пермисион.МОДЕРАТЕ) деф
модерате_ена ле(ид): коммент =
    Цоммент.куери.гет_оп_404(ид)
    коммент.диса лед = Нетачно
    д .сессион.адд(коммент)
    ретурн редирект(урл_фор('модерате',
        паге=рекуест.аргс.гет('паге', 1, типе=инт)))}

@маин.роуте('/модерате/диса ле/<инт:ид>')
@логин_рекуиред @
пермисион_рекуиред(Пермисион.МОДЕРАТЕ) деф
модерате_диса ле(ид):
    коммент = Цоммент.куери.гет_оп_404(ид)
    коммент.диса лед = Тачно
    д .сессион.адд(коммент)
    ретурн редирект(урл_фор('модерате',
        паге=рекуест.аргс.гет('паге', 1, типе=инт)))}
```

Руте за омогућавање и онемогућавање коментара учитавају о јекат коментара, постављају онемогућено поље на одговарајућу вредност и записују га назад у азу података. На крају, они преусмеравају назад на страницу за модерирање коментара (приказана на [слици 13-3](#)), и ако је аргумент странице дат у стрингу упита, они га укључују у преусмеравање. Дугмад у ша лону \_цомментс.хтмл се приказују са аргументом странице тако да преусмеравање враћа кориснику на исту страницу.



Слика 13-3. Страница за модерирање коментара



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 13 да исте проверили ову верзију апликације.

Тема друштвених оележја је употребљена овим поглављем. У следећем поглављу ћете научити како да изложите функционалност апликације као АПИ који клијенти као што су апликације за паметне телефоне могу да користе.

## ГЛАВА 14

# Интерфејси за програмирање апликација

Последњих година у већини апликацијама постоји тренд да се све више пословне логике премести на страну клијента, производећи архитектуру која је позната као Рицх Интернет Апликационс (РИА). У РИА, главна (а понекад и једини) функција сервера је да клијентској апликацији пружи услуге преузимања и складиштења података. У овом моделу, сервер постаје већ сервис или интерфејс за програмирање апликација (АПИ).

Постоји неколико протокола помоћу којих РИА могу да комуницирају са већином услугом. Протоколи за даљински позив процедуре (РПЦ) као што је КСМЛ-РПЦ или његов дериват, Симплifiед Ојеџт Аццес Протоцол (СОАП), или су популарни извор пре неколико година. У скорије време, архитектура Репрезентативан Стате Трансфер (РЕСТ) се појавила као фаворит за већине апликације због тога што је изграђена на познатом моделу Ворлд Виде Веба.

Флакс је идеалан оквир за изградњу РЕСТфул веб-сервиса, захваљујући својој лаганој природи. У овом поглављу ћете научити како да имплементирате РЕСТфул АПИ заснован на Флаксу.

## Увод у РЕСТ

[Докторска дисертација](#) Роја Филдинга описује РЕСТ архитектонски стил за већине сервисе у смислу његових шест карактеристика:

Клијент-сервер

Мора постојати јасна подела између клијената и сервера.

Апаратид

Захтев клијента мора да садржи све информације које су неопходне за његово извршење. Сервер не сме да складиши стање о клијенту које траје од једног захтева до другог.

## Кеш

одговори са сервера могу ити означени као кеширани или некеширани тако да клијенти (или посредници између клијената и сервера) могу да користе кеш меморију у сврху оптимизације.

## Јединствени интерфејс

Протокол којим клијенти приступају серверским ресурсима мора ити доследан, до ро дефинисан и стандардизован. Ово је најкомплекснији аспект РЕСТ-а, који покрива употребу јединствених идентификатора ресурса, представљања ресурса, самоописних порука између клијента и сервера и хипермедије.

## Слојевити

системски прокси сервери, кеш меморије или мрежни пролази могу се уметнути између клијената и сервера по потребе и да и се по ољшале перформансе, поузданост и скала илност.

## Код на захтев Клијенти

могу опционо да преузму код са сервера да и га извршили у свом контексту.

## Ресурси су све Концепт ресурса је срж

РЕСТ архитектонског стила. У овом контексту, ресурс је ставка од интереса у домену апликације. На пример, у апликацији за логовање, корисници, постови на логу и коментари су сви ресурси.

Сваки ресурс мора имати јединствени идентификатор који га представља. Када радите са ХТТП-ом, идентификатори ресурса су УРЛ адресе. Настављајући са примером логовања, пост на логу може ити представљен УРЛ-ом /апи/постс/12345, где је 12345 идентификатор ојаве, као што је примарни кључ азе података поста. Формат или садржај УРЛ адресе заправо нису итни; све што је важно је да сваки УРЛ ресурс јединствено идентификује ресурс.

Колекција свих ресурса у класи такође има додељену УРЛ адресу. УРЛ за колекцију лог постова може ити /апи/постс/, а УРЛ за прикупљање свих коментара може ити /апи/цомментс/.

АПИ такође може да дефинише УРЛ адресе колекције које представљају логичке подскупове свих ресурса у класи. На пример, зирка свих коментара у лог посту 12345 може ити представљена УРЛ-ом /апи/постс/12345/цомментс/. Уо ичайено је дефинисати УРЛ адресе које представљају колекције ресурса са косом цртом на крају, јер им то даје репрезентацију „поддиректоријума“.



Имајте на уму да Флакс примењује посебан третман на руте које се завршавају косом цртом. Ако клијент затражи УРЛ без завршне косе црте и постоји одговарајућа ruta која има косу црту на крају, онда ће Флакс аутоматски одговорити преусмеравањем на УРЛ завршну косу црту. За оврнуту случају се не издају преусмеравања.

## Рекуест Метододс

Клијентска апликација шаље захтеве серверу на утврђене УРЛ адресе ресурса и користи метод захтева да укаже на жељену операцију. Да исте до или листу доступне постове на логу у АПИ-ју за логовање, клијент и послала ГЕТ захтев на `http://www.екампле.ком/апи/постс/`, а да исте уметнули нови лог пост, то и послало ПОСТ захтев на исту УРЛ адресу, са садржајем лог поста у телу захтева. До преузми лог пост 12345 клијент и послала ГЕТ захтев на `http://www.екампле.ком/апи/постс/12345`. Та ела 14-1 наводи методе захтева које се оично користе у РЕСТфул АПИ-јима, са њиховим значењима.

Та ела 14-1. Методе ХТТП захтева у РЕСТфул АПИ-јима

Захтев методом	Таргет	Опис	ХТТП одговор статусни код
добрити	УРЛ појединачног ресурса На авите ресурс.		200
добрити	УРЛ за прикупљање ресурса На авите колекцију ресурса (или једну страницу из ње ако је сервер имплементира пагинацију).		200
пошта	УРЛ колекције ресурса Креирајте нови ресурс и додајте га у колекцију. Сервер ира УРЛ новог ресурса и враћа га у заглавље локације у одговору.		201
ставити	УРЛ појединачног ресурса Измените постојећи ресурс. Алтернативно, овај метод такође може користити за креирање новог ресурса када клијент може да изајре УРЛ ресурса.		200 или 204
избираши	УРЛ појединачног ресурса Избриши ресурс.		200 или 204
избираши	УРЛ зирке ресурса Избриши све ресурсе у колекцији.		200 или 204



РЕСТ архитектура не захтева да све методе буду имплементиране, поменуто за ресурс. Ако клијент позове метод који није подржано за дати ресурс, затим одговор са статусом 405 код (метод није дозвољен) треба вратити. Флакс се ави овим грешка аутоматски.

**Методе захтева ГЕТ, ПОСТ, ПУТ и ДЕЛЕТЕ нису једине. ХТТП протокол се ослања на друге методе, као што су ХЕАД и ОПТИОНС, које су аутоматски имплементирао Флакс.**

## Органи за тражење и одговор

Ресурси се шаљу напред и назад између клијента и сервера у телима захтева и одговоре, али РЕСТ не наводи формат који ће се користити за кодирање ресурса. Технологија типа садржаја у захтевима и одговорима се користи за означавање формата у који је ресурс кодиран у телу. Стандардни механизам преговарања о садржају

Низми у ХТТП протоколу се могу користити између клијента и сервера да и се договорили о формату који о а подржавају.

Два формата која се о ично користе са РЕСТфул ве услугама су ЈаваСкрипт О јец Нотатион (JSON) и Ектенси ле Маркуп Лангуаге (КСМЛ). За РИА засноване на ве у, JSON је привлачен з ог тога што је много сажетији од КСМЛ-а и з ог лиских веза са ЈаваСкриптом, скрипт језиком на страни клијента који користе ве прегледачи.

Враћајући се на АПИ пример лога, ресурс поступка на логу може имати следећу JSON репрезентацију:

```
{
    "селф_урл": "хттп://www.екампле.цом/апи/постс/12345", "титле":
    "Писање РЕСТфул АПИ-ја у Питхону", "аутхор_урл": "хттп ://
    www.екампле.цом/апи /серс/2", " оди": "... текст чланка
    овде ...", "комментс_урл": "хттп://www.екампле.цом/апи/постс/
    12345/комментс"
}
```

О ратите пажњу на то како су поља урл, аутхор\_урл и коментс\_урл потпуно квалифициране УРЛ адресе ресурса. Ово је важно јер ови УРЛ-ови омогућавају клијенту да открије нове ресурсе. цес.

У до ро дизајнираном РЕСТфул АПИ-ју, клијент зна кратку листу УРЛ адреса ресурса највишег нивоа, а затим открива остатак из веза укључених у одговоре, слично као што можете да откријете нове ве странице док претражујете ве кликом на везе које се појављују на страницама за које знате.

### Версионирање

У традиционалној ве апликацији усмереној на сервер, сервер има потпуну контролу над апликацијом. Када се апликација ажурира, инсталирање нове верзије на сервер је доволно да се ажурирају сви корисници јер се чак и делови апликације који се покрећу у корисниковом ве претраживачу преузимају са сервера.

Ситуација са РИА-има и ве услугама је компликованија, јер се клијенти често развијају независно од сервера — можда чак и од стране различитих људи. Размотрите случај апликације у којој РЕСТфул ве услугу користе различити клијенти, укључујући ве претраживаче и матичне клијенте паметних телефона. Клијент ве претраживача може да се ажурира на серверу у ило ком тренутку, али апликације за паметне телефоне не могу да се ажурирају на силу; власник паметног телефона тре а да дозволи ажурирање. Чак и ако је власник паметног телефона вољан да се ажурира, није могуће организовати надоградњу свих постојећих инстанци апликација за паметне телефоне тако да се тачно поклопе са применом нове верзије сервера.

Из ових разлога, ве сервиси морају да уду толерантнији од о ичних ве апликација и да уду у стању да раде са старим верзијама својих клијената. Промене на ве -услуги морају да се врше са изузетном пажњом, јер промене некомпати илне уназад могу да изазову

постојеће клијенте да прекину док се не надограде. Уо ичайена пракса је да се ве сервисима да верзија, која се додаје свим УРЛ-овима дефинисаним у тој верзији серверске апликације. На пример, прво издање ве сервиса за логовање могло и да открије колекцију лог постова на /апи/в1/постс/.

Укључивање верзије ве услуге у УРЛ помаже да стварне и нове функције уду организоване тако да сервер може да пружи нове функције новим клијентима док наставља да подржава стварне клијенте. Ажурирање сервиса за логовање могло и да промени JSON формат постова на логу и сада ојави постове на логу као /апи/в2/постс/, уз задржавање старијег JSON формата за клијенте који се повезују на /апи/в1/постс/.

Иако подршка за више верзија сервера може постати оптерећење одржавања, постоје ситуације у којима је то једини начин да се дозволи развој апликације изазивања пролема постојећим имплементацијама. Старије верзије услуге могу исти застареле и касније уклоњене када сви клијенти пређу на новију верзију.

## РЕСТфул ве услуге са фласком

Флакс чини веома лаким креирање РЕСТфул ве услуга. Познати декоратор роуте() заједно са опционим аргументом његових метода може се користити за декларисање пута које рукују УРЛ-овима ресурса које је услуга изложила. Рад са JSON подацима је такође једноставан, јер се JSON подаци укључени у захтев могу исти у формату речника позивањем `reuest.get_json()`, а одговор који трећа да садржи JSON може се лако генерисати из Питхон речника користећи `Flask-ов jsonify()` помоћна функција.

Следећи одељци показују како се Флакски може проширити помоћу РЕСТфул ве услуге која клијентима даје приступ постовима на логу и сродним ресурсима.

### Креирање АПИ нацрта Руте

повезане са РЕСТфул АПИ-јем чине самостални подскуп апликације, тако да је њихово стављање у сопствени нацрт најбољи начин да их дођете организујете. Општа структура АПИ нацрта унутар апликације приказана је у [Примеру 14-1](#).

### Пример 14-1. АПИ нацрт структуре

```
|-флакси
 |-апп/
   |-апи
     |__инит_.пи |
     |серв.пи |
     |пост.пи |
     |коммент.пи |
     |аутентицијон.пи |
     |эррор.пи |
     |декоратор.пи
```

Одјратите пажњу на то како пакет који се користи за АПИ укључује број верзије у свом називу. Ако у будућности буде потребно увести верзију АПИ-ја која није компатибилна уназад, она се може додати као посебан пакет са различитим бројем верзије и ова АПИ-ја могући ити укључена у апликацији.

АПИ нацрт имплементира сваки ресурс у посебан модул. Модули који воде рачуна о аутентификацији и руководе грешкама и да ове езеде прилагођене декораторе су такође укључени. Конструктор нацрта је приказан у [примеру 14-2](#).

Пример 14-2. `апп/апи/_инит_.пи:` Креирање АПИ нацрта

`из флајск импорт Блуепринт`

`апи = Блуепринт('апи', __наме__)`

`од . аутентификацију увоза , постове, кориснике, коментаре, грешке`

Структура конструктора пакета нацрта је слична оној код осталих нацрта. Увоз свих компоненти нацрта је неопходан да и се регистровале руте и други руководоци. Пошто многи од ових модула морају да увезу нацрт АПИ-ја који је овде референциран, увоз се врши на дну како и се спречиле грешке због кружних зависности.

Регистрација АПИ нацрта је приказана у [Примеру 14-3](#).

Пример 14-3. `апп/инит.пи:` Регистрација нацрта АПИ-ја

```
деф цреате_апп(цонфиг_наме):
    #
    # ...
    из .апи импорт апи као апи_
    луепринт
    апи.регистер_ луепринт(апи_, луепринт, урл_префикс='/апи/в1') #
    ...
    ...
```

АПИ нацрт је регистрован са УРЛ префиксом, тако да ће све његове руте имати своје УРЛ адресе са префиксом /апи/в1. Додавање префикса приликом регистрације нацрта је до паре идеја јер елиминише потребу за чврстим кодирањем броја верзије у сваком нацрту рута.

Руковање грешкама

РЕСТфул веб услуга озвештава клијента о статусу захтева слањем одговарајућег ХТТП статусног кода у одговору, плус све додатне информације у телу одговора. Типични статусни кодови које клијент може очекивати од веб сервиса наведени су у [табели 14-2](#).

Та ела 14-2. Кодови статуса ХТТП одговора које о ично враћају АПИ-ји

ХТТП статус код	Име	Опис
200	У раду	Захтев је успешно завршен.
201	Цреатед	Захтев је успешно завршен и као резултат је креiran нови ресурс.
202	Прихваћено	Захтев је прихваћен на о раду, али је још увек у току и иће покренут асинхроно.
204	Без садржаја	Захтев је успешно завршен и нема података за враћање у одговору.
400	Лош захтев	Захтев је неважећи или недоследан.
401	Неовлашћено	Захтев не укључује информације о аутентификацији или су наведени акредитиви неважећим.
403	За рађено	Акредитиви за аутентификацију послити уз захтев нису довољни за захтев.
404	Није пронађен	Ресурс наведен у УРЛ-у није пронађен.
405		Метод није дозвољен. Тражени метод није подржан за дати ресурс.
500		Интерна грешка сервера Дошло је до неочекиване грешке током о раде захтева.

Руковање статусним кодовима 404 и 500 представља малу компликацију у томе што ови грешке о ично генерише Флакс сам и враћа ХТМЛ одговор. Ово може з унити АПИ клијента, који ће вероватно очекивати све одговоре ЈСОН формат.

Један од начина да се генеришу одговарајући одговори за све клијенте је да се грешка хандлери прилагођавају своје одговоре на основу формата који захтева клијент, техника названо преговарање о садржају. Пример 14-4 показује по ољшани о рађивач грешака 404 који одговара са ЈСОН клијентима ве услуга и ХТМЛ-ом за друге. Грешка 500 хандлер је написан на сличан начин.

Пример 14-4. апп/апи/еррорхандлер(404): 404 о рађивач грешака са ХТТП преговарањем садржаја

```
@майн.апп_еррорхандлер(404)
деф паге_нот_фоунд(е):
    ако рекуест.аццепт_миметипес.аццепт_јсон и \
        не рекуест.аццепт_миметипес.аццепт_хтмл:
        одговор = јсонифи({'еррор': 'није пронађено'})
        одговор.статус_цоде = 404
        повратни одговор
        ретурн рендер_темплате('404.хтмл'), 404
```

Ова нова верзија руковаоца грешкама проверава заглавље захтева за прихватање, што је декодирано у рекуест.аццепт\_миметипес, да и се одредило који формат клијент жели одговор у. Претраживачи генерално не наводе никаква ограничења за одговор за простирике, али АПИ клијенти о ично раде. ЈСОН одговор се генерише само за клијенте који укључују ЈСОН на своју листу прихваћених формата, али не и ХТМЛ.

Преостале статусне кодове експлицитно генерише ве услуга, тако да се могу имплементирати као помоћне функције унутар нацрта у модулу errorc.pi.

**Пример 14-5** показује имплементацију грешке 403; остали су слични.

Пример 14-5. апп/апи/errorc.pi: АПИ за руковање грешкама за статусни код 403

деф за рањено(порука):

```
респонсе = jsonify({'error': 'фор иден', 'message': message})
респонсе.статус_код = 403 врати одговор
```

Функције приказа у нацрту АПИ-ја могу позвати ове помоћне функције за генерисање одговора на грешке када је то потребе но.

### Аутентификација корисника помоћу Флакс-ХТТПАутх

Ве услуге, као и о ичне ве апликације, тре а да штите информације и о ез еде да се оне не дају неовлашћеним странама. Из тог разлога, РИА морају тражити од својих корисника акредитиве за пријаву и проследити их серверу ради верификације.

Раније је поменуто да је једна од карактеристика RECTful ве сервиса да су ез држављанства, што значи да серверу није дозвољено да „памти“ ило шта о клијенту између захтева. Клијенти морају у самом захтеву да наведу све информације неопходне за извршење захтева, тако да сви захтеви морају да садрже корисничке акредитиве.

Тренутна функционалност пријављивања имплементирана уз помоћ Флакс-Логин-а складишти податке у корисничкој сесији, које Флакс подразумевано складиши у колачић на страни клијента, тако да сервер не складиши никакве информације везане за корисника; тражи од клијента да га уместо тога сачува. Чини се да је ова имплементација у складу са захтевом RECT-а ез држављанства, али употреба колачића у RECTful ве сервисима спада у сиву зону, јер може ити гломазно за клијенте који нису ве претраживачи да их имплементирају. Из тог разлога, коришћење колачића у АПИ-јима се генерално сматра лошим из ором дизајна.



Захтев за RECT ез држављанства може изгледати престрог, или није произвољан. Сервери ез државности могу врло лако да се скалирају. Ако сервери чувају информације о клијентима, потребе но је о ез едити да исти сервер увек до ија захтеве од датог клијента, или у супротном користити дељено складиште за податке клијента. О а су сложена про лема за решавање који не постоје када је сервер ез држављанства.

Пошто је RECTful архитектура заснована на ХТТП протоколу, ХТТП аутентификација је префериирани метод који се користи за слање акредитива, ило у основном или Дигест стилу. Са ХТТП аутентификацијом, кориснички акредитиви су укључени у заглавље ауторизације са свим захтевима.

ХТТП протокол аутентификације је довољно једноставан да се може директно имплементирати, или проширење Флакс-ХТТПАутх пружа погодан омотач који сакрива детаље протокола у декоратору сличном Флакс-Логин-овом логин\_рекуриред.

Флакс-ХТТПАутх се инсталира са пип-ом:

```
(венв) $ pip install flask-httplibx
```

Да исте иницијализовали проширење за ХТТП Басиц аутентификацију, мора се креирати о јекат класе ХТТПБасицАутх . Као и Флакс-Логин, Флакс-ХТТПАутх не прави никакве претпоставке о процедуре која је потребна за верификацију корисничких акредитива, тако да се ове информације дају у функцији повратног позива. [Пример 14-6](#) показује како се локални иницијализује и до ија повратни позив за верификацију.

**Пример 14-6. апп/апи/аутхентицацијон.пи: Флакс-ХТТПАутх иницијализација**

```
фром flask_httplibx импорт ХТТПБасицАутх аутх =
ХТТПБасицАутх()
```

```
@аутх.верифи_пассворд деф
верифи_пассворд(е-пошта, лозинка): иф email ==
    "": ретурн Фалсе user =
        User.куери.фильтер_ и(емаил =
            email).firstr() ако није корисник:

    ретурн Фалсе
г.цуррент_усер = корисник
враћа user.verifi_password(пассворд)
```

Пошто ће се овај тип аутентификације корисника користити само у нацрту АПИ-ја, проширење Флакс-ХТТПАутх се иницијализује у пакету нацрта, а не у пакету апликације као друга проширења.

Е-пошта и лозинка су верификовани коришћењем постојеће подршке у корисничком моделу. Повратни позив за верификацију враћа Труе када је пријава важећа и Фалсе у супротном. Екstenзија Флакс-ХТТПАутх ће такође позвати повратни позив за захтеве који не носе аутентификацију, постављајући о а аргумента на празан стринг. У овом случају, када је email празан стринг, функција одмах враћа Фалсе да локира захтев; за одређене апликације може ити прихватљиво дозволити анонимном кориснику враћањем Труе. Повратни позив за аутентификацију чува аутентификованог корисника у Флаксовој променљивој контекста г тако да функција прегледа може да јој приступи касније.



Пошто се кориснички акредитиви размењују са сваким захтевом, изузетно је важно да АПИ руте буду изложене преко ез једног ХТТП-а како и сви захтеви и одговори или шифровани у транзиту.

Када су акредитиви за аутентификацију неважећи, сервер враћа клијенту одговор 401 статусног кода. Флакс-ХТТПАутх подразумевано генерише одговор са овим статусним кодом, али да и се осигурало да је одговор конзистентан са другим грешкама које враћа АПИ, одговор на грешку се може прилагодити као што је приказано у [Примеру 14-7.](#)

Пример 14-7. \_апп/апи/аутхентицацијон.пи: О рађивач грешака Флакс-ХТТПАутх

из .еррорс увоз неовлашћен

```
@аутх.еррор_хандлер
деф аутх_еррор():
    врати неовлашћено('Неважећи акредитиви')
```

За заштиту руте користи се декоратор аутх.логин\_рекуиред :

```
@апи.роуте('/постс/')
@аутх.логин_рекуиред
деф get_postс(): пролаз
```

Али пошто све руте у нацрту морају или заштићене на исти начин, декоратор логин\_рекуиред може или укључен једном у о рађивач ефоре\_рекуест за нацрт, као што је приказано у [Примеру 14-8.](#)

Пример 14-8. апп/апи/аутхентицацијон.пи: о рађивач ефоре\_рекуест са аутентификацијом

из .еррорс увоз за рањен

```
@апи. ефоре_рекуест
@аутх.логин_рекуиред
деф ефоре_рекуест():
    ако није г.циррент_усер.ис_анонимоус и \ није
        г.циррент_усер.ценфирмед:
            врати фор идден('Непотврђени налог')
ретурн фор идден('Непотврђени налог')
```

Сада ће се провере аутентификације о ављати автоматски за све руте у нацрту. Као додатна провера, о рађивач ефоре\_рекуест такође од ија проверене кориснике који нису потврдили своје налоге.

Аутентификација заснована на токенима

Клијенти морају да пошаљу акредитиве за аутентификацију уз сваки захтев. Да исте из егли стално преношење осетљивих информација као што је лозинка, може се користити решење за аутентификацију засновано на токенима.

У аутентификацији заснованој на токенима, клијент захтева приступни токен слањем захтева који укључује акредитиве за пријаву као аутентификацију. Токен се тада може користити уместо акредитива за пријаву за аутентификацију захтева. Из ез једносних разлога,

токени се издају са припадајућим истеком. Када истекне токен, клијент мора поново да се аутентификује да и до ио нови. Ризик да жетон дође у погрешне руке је ограничен з ог његовог кратког века трајања. [Пример 14-9](#) показује две нове методе додате моделу корисника које подржавају генерисање и верификацију токена за аутентификацију коришћењем његовог опасног.

Пример 14-9. апп/моделс.пи: подршка за аутентификацију засновану на токену

```
класс Усер(д .Модел): # деф
    генерате_аутх_токен(селф,
        експирацион):
        с = серијализатор(цуррент_апп.ционфиг['СЕЦРЕТ_КЕИ'],
            експрес_ин=експирацион)
        ретурн с.думпс({'ид': селф.ид}).децоде('утф-8')

    @статичметод
    деф верифи_аутх_токен(токен): с =
        Серијализатор(цуррент_апп.ционфиг['СЕЦРЕТ_КЕИ']) покушај: подаци
        = с.лоадс(токен)

    осим:
        врати Ништа
    ретурн Усер.куери.get(дата['ид'])


```

Генерирати\_аутх\_токен () метода враћа потписани токен који кодира поље ИД -а корисника . Такође се користи време истека у секундама. Метода верифи\_аутх\_токен() узима токен и, ако се утврди да је исправан, враћа кориснику ускладиштеног у њему. Ово је статична метода, јер ће корисник или познат тек након што се токен декодира.

Да исте потврдили аутентичност захтева који долазе са токеном, верифи\_пассворд повратни позив за Флакс-ХТТПАутх мора или модификован тако да прихвати токене као и редовне акредитиве. Ажурирани повратни позив је приказан у [примеру 14-10](#).

Пример 14-10. апп/апи/аутхентицијацион.пи: по ољшана верификација аутентификације са подршком за токене

```
@аутх.верифи_пассворд деф
верифи_пассворд(емайл_ор_токен, лозинка):
    ако email_or_token == '':
        ретурн Фалсе иф
    парсворд == '':
        г.цуррент_усер = Усер.верифи_аутх_токен(емайл_ор_токен) г.токен_усед =
            Тачан повратак г.цуррент_усер није Ноне усер =
                Усер.куери.филтер_ и(емайл=емайл_ор_токен).фирст() ако није корисник :
                    ретурн Фалсе
                г.цуррент_усер = корисник
```

```
г.токен_усед = лажно
враћање user.verifi_password(пассворд)
```

У овој новој верзији, први аргумент за аутентификацију може ити адреса е-поште или токен за аутентификацију. Ако је ово поље празно, претпоставља се да је анонимни корисник, као и раније. Ако је лозинка празна, тада се претпоставља да је поље email\_or\_token токен и као такво је потврђено. Ако о а поља нису празна, онда се претпоставља регуларна аутентификација путем е-поште и лозинке. Са овом имплементацијом, аутентификација заснована на токенима је опциона; на сваком клијенту је да га користи или не. Да и функцијама приказа дала могућност разликовања између два метода аутентификације, додата је променљива г.токен\_усед.

Рута која враћа токене за аутентификацију клијенту је такође додата нацрту АПИ-ја. Имплементација је приказана у [примеру 14-11](#).

Пример 14-11. апп/апи/аутхентицацијон.пи: генерисање токена за аутентификацију

```
@апи.роуте('/токенс/', методс=['ПОСТ']) деф
рет_токен(): ако је г.цуррент_усер.ис_анонимоус
или г.токен_усед: ретурн унаутхоризед('Неважећи
акредитиви') ретурн јсонифи({'токен':
г.цуррент_усер.генерате_аутх_токен(
екпиратион=3600), 'екпиратион': 3600})
```

Пошто је ова ruta у нацрту, механизми аутентификације додати руковаоцу ефоре\_рекуест такође се примењују на њу. Да и спречили клијенте да се аутентификују на ову руту користећи претходно до ијени токен уместо адресе е-поште и лозинке, проверава се променљива г.токен\_усед, а захтеви оверени токеном се од ијају. Сврха овога је да спречи кориснике да зао иђу истек токена тако што ће захтевати нови токен користећи стари токен као аутентификацију. Функција враћа токен у JSON одговору са периодом важења од једног сата. Тачка је такође укључена у JSON одговор.

Серијализација ресурса у и из JSON -а Честа потре а

при писању ве услуге је конвертовање интерних репрезентација ресурса у и из JSON-а, што је транспортни формат који се користи у ХТТП захтевима и одговорима. Процес претварања интерне репрезентације у транспортни формат као што је JSON назива се серијализација. [Пример 14-12](#) показује нову методу то\_јсон() додату класи Пост .

Пример 14-12. апп/модел.пи: претварање поста у JSON речник који може да се серијализује

```
класс Пост(д .Модел):
# ...
дeф тo_жсон(сeлф):
жсон_пoст = {
    'урл': урл_фор('апи.гет_пoст', ид=сeлф.ид), 'оди':
        сeлф.оди, 'оди_хтмл': сeлф.оди_хтмл,
    'тиместамп': сeлф.тиместамп, 'aутхор_урл':
        урл_фор('апи.гет_усер', ид=сeлф.aутхор_ид),
    'цомментc_урл': урл_фор('апи.гет_пoст_цомментc', ид=сeлф.ид),
    'цоммент_цоунт': сeлф.цомментc.цоунт()

} рeтурн жсон_пoст
```

Поља урл, аутхор\_урл и цомментc\_урл морају да врате УРЛ адресе за одговарајуће ресурсе, тако да се генеришу позивима урл\_фор() ка другим путама које ће бити дефинисане у нацрту АПИ-ја.

Овај пример показује како је могуће вратити „измишљене“ атрибуте у представљању ресурса. Поље цоммент\_цоунт враћа број коментара који постоје за пост на логу. Иако ово није стварни атрибут модела, он је укључен у репрезентацију ресурса као погодност за клијента.

Метода тo\_жсон() за корисничке моделе може се конструисати на сличан начин. Овај метод је приказан у примеру 14-13.

Пример 14-13. апп/модел.пи: претварање корисника у JSON речник који може да се серијализује

```
класс Усер(УсерМикин, д .Модел):
# ...
дeф тo_жсон(сeлф):
жсон_усер = {
    'урл': урл_фор('апи.гет_усер', ид=сeлф.ид),
    'усернaмe': сeлф.усернaмe, 'мем' ер_синце':
        сeлф.мем.ер_синце, 'ласт_сеен': сeлф.ласт_сеен,
    'пoстc_урл': урл_фор('апи.гет_усер_пoстc',
        ид=сeлф.ид), 'фoлловeд_пoстc_урл':
        урл_фор('апи.гет_усер_фoлловeд_пoстc', ид=сeлф.ид), 'пoст_цоунт':
            сeлф.пoстc.цоунт()

} рeтурн жсон_усер
```

Односите пажњу на то како су у овој методи неки од атрибута корисника, као што су е-пошта и улога, изостављени из одговора из разлога приватности. Овај пример поново показује

да репрезентација ресурса који се нуди клијентима не мора ити идентична интерној дефиницији одговарајућег модела азе података.

Инверзна серијализација се назива десеријализација. Десеријализација ЈСОН структуре назад у модел представља изазов да неки подаци који долазе од клијента могу ити неважећи, погрешни или непотре ни. [Пример 14-14](#) показује метод који креира пост из ЈСОН-а.

Пример 14-14. апп/моделс.пи: креирање поста на логу из ЈСОН-а

[из апп.екцептионс импорт ВалидационЕррор](#)

```
класс Пост(д .Модел):
    # @статицметод
    деф
        фром_јсон(јсон_пост):
            оди
                = јсон_пост.рет(' оди') ако је
                тело Нема или тело == "": подизање
                    ВалидационЕррор('пост нема тело') повратни
                    пост(тело=тело)
```

Као што видите, ова имплементација ира да користи само атриут тела из ЈСОН речника. Атриут оди\_хтмл се занемарује пошто се рендеровање Маркдовн на страни сервера аутоматски покреће СКЛАЛЦХЕМИ догађајем сваки пут када се модификује атриут тела . Атриут временске ознаке не мора да се даје осим ако клијенту није дозвољено да о јављује постове са датумом уназад или у удућности, што није функција коју ова апликација подржава. Поље аутхор\_урл се не користи јер клијент нема овлашћења да изајре аутора лог поста; једина могућа вредност за ово поље је вредност аутентификованог корисника. Атриут цомментс\_урл и цоммент\_џоунт се аутоматски генеришу из односа азе података, тако да у њима нема корисничких информација које су потребне за креирање ојаве. Коначно, урл поље се занемарује јер у овој имплементацији УРЛ-ове ресурсе дефинише сервер, а не клијент.

Оратите пажњу на то како се врши провера грешака. Ако поље тела недостаје или је празно, онда се покреће изузетак ВалидационЕррор . Подизање изузетка је у овом случају прикладан начин за решавање грешке, јер овај метод нема довољно знања да правилно оради услов грешке. Изузетак ефективно прослеђује грешку позиваоцу, омогућавајући коду вишег нивоа да оради грешку. Класа ВалидационЕррор је имплементирана као једноставна поткласа Пајтонове ВалуеЕррор.

Ова имплементација је приказана у [Примеру 14-15](#).

Пример 14-15. апп/екцептионс.пи: ВалидационЕррор изузетак

[класа ВалидационЕррор\(ВалуеЕррор\):](#)

проћи

Апликација сада трећа да о ради овај изузетак пружањем одговарајућег одговора клијенту. Да исте из егли додавање кода за хватање изузетака у функције приказа, глобални руковалац изузетком се може инсталарирали помоћу Флаксоговог декоратора за о раду грешака. Руковалац за изузетак ВалидациоnError је приказан у примеру 14-16.

Пример 14-16. апп/апи/еррорс.пи: АПИ за руковање грешкама за изузетке ВалидациоnError

```
@апи.еррорхандлер(ВалидациоnError)
деф валидациоn_еррор(е): ретурн
    ад_рекуест(е.арг[0])
```

Декоратор за о раду грешака је исти онај који се користи за регистровање руковалаца за ХТТП статусне кодове, али у овој употреби узима класу Екцептион као аргумент. Декорисана функција ће ити позвана сваки пут када се подигне изузетак дате класе.

Имајте на уму да се декоратор доји из АПИ нацрта, тако да ће овај руковалац ити позван само када се подигне изузетак док се рукује рутом из нацрта. Користећи ову технику, функције кода у приказу могу ити написане врло јасно и концизно, јез потреће да се укључи провера грешака. На пример:

```
@апи.роуте('/постс/', методс=['ПОСТ']) деф
нев_пост(): пост = Пост.фром_јсон(рекуест.јсон)
    пост.аутхор = г.цуррент_усер
    д.сессион.адд(пост) д.сессион.цоммит()
    ретурн јсонифи(пост.то_јсон())
```

### Имплементација крајњих тачака ресурса Остаје

да се имплементирају руте које рукују различитим ресурсима. ГЕТ захтеви су оично најлакши јер само враћају информације и не морају да уносе никакве измене. Пример 14-17 показује два ГЕТ руковаоца за постове на логу.

Пример 14-17. апп/апи/постс.пи: ГЕТ руковаоце ресурсима за постове

```
@апи.роуте('/постс/')
деф гет_постс(): постови
    = Пост.куери.алл() ретурн
        јсонифи({ 'постс': [пост.то_јсон() за пост у постовима] })

@апи.роуте('/постс/<инт:ид>') деф
гет_пост(ид): пост =
    Пост.куери.гет_оп_404(ид) ретурн
        јсонифи(пост.то_јсон())
```

Прва рута орађује захтев за прикупљање постова. Ова функција користи разумевање листе за генерирање ЈСОН верзије свих постова. Други пут

враћа један пост на логу и одговара грешком кода 404 када дати ИД није пронађен у ази података.

ПОСТ о рађивач за ресурсе лог постова у ацује нови лог пост у азу података. Овај пут је приказан у [примеру 14-18](#).

Пример 14-18. апп/апи/постс.пи: ПОСТ о рађивач ресурса за постове

```
@апи.роуте('/постс/', методс=['ПОСТ'])
@пермиссион_рекуиред(Пермиссион.ВРИТЕ)
деф нев_пост(): пост = Пост.фром_јсон(рекуест.јсон)
    пост.аутхор = г.цуррент_усер д .
    сессион.адд(пост) д .сессион.цоммит()
    ретурн јсонифи(пост.то_јсон()), 201, \
        {'Лоцијон': урл_фор('апи.гет_пост',
        ид=пост.ид)}
```

Ова функција приказа је умотана у декоратор пермиссион\_рекуиред (приказано у следећем примеру) који овде је јеђује да аутентификовани корисник има дозволу да пише постове на логу. Стварно креирање лога поста је једноставно због подршке за руковање грешкама која је претходно имплементирана. Оваја на логу је направљена од JSON података и њен аутор је експлицитно додељен као аутентификовани корисник. Након што је модел уписан у азу података, враћа се статусни код 201 и додаје се заглавље локације са УРЛ-ом новокреiranог ресурса.

Имајте на уму да као погодност за клијенте тело одговора укључује нови ресурс. Ово ће спасити клијента од потребе да за њега изда ГЕТ захтев одмах након креирања ресурса.

Пермиссион\_рекуиред декоратор који се користи да спречи неовлашћене кориснике да креирају нове постове на логу сличан је оном који се користи у апликацији, али је прилагођен за нацрт АПИ-ја. Имплементација је приказана у [Примеру 14-19](#).

Пример 14-19. апп/апи/децораторс.пи: пермиссион\_рекуиред декоратор

```
деф пермиссион_рекуиред(пермиссион):
    деф декоратор(ф): @врапс(ф) деф
        децоратед_функцион(*аргс,
        **кваргс):
            ако није г.цуррент_усер.цан(пермиссион):
                ретурн фор иdden('Недовољне дозволе')
            ретурн ф(*аргс, **кваргс)
        ретурн децоратед_функцион
    ретурн декоратор
```

Приказан је ПУТ о рађивач за постове на логу, који се користи за уређивање постојећих ресурса  
**Пример 14-20.**

Пример 14-20. ап/апи/постс.пи: ПУТ о рађивач ресурса за постове

```
@апи.роуте('/постс/<инт:ид>', методс=['PUT'])
@пермиссион_рекуриред(Пермиссион.ВРИТЕ)
деф едит_пост(ид):
    пост = Пост.куери.get_op_404(ид)
    ако г.цуррент_усер != пост.аутхор и \
        не г.цуррент_усер.цан(Пермиссион.АДМИН):
        врати фор иден('Недовољне дозволе')
    пост.оди = реквест.json.get('оди', пост.оди)
    д .сессион.адд(пост)
    д .сессион.цоммит()
    врати jsonифи(пост.to_json())
```

Провере дозвола су у овом случају сложеније. Стандардна провера за по мисија писања постова на логу се оавља са декоратором, али да и се омогутило кориснику да уређује а лог пост функција такође мора да озеди да је корисник аутор поста или у супротном је администратор. Ова провера се експлицитно додаје функцији приказа. Ако ова провера морало се додати у многе функције приказа, израда декоратора за то и ила до ра начин да се избегне понављање кода.

Пошто апликација не дозвољава писање постова, руковаца за ДЕЛЕТЕ метод захтева не мора ити имплементиран.

О рађивачи ресурса за кориснике и коментаре имплементирани су на сличан начин. Та ела 14-3 наводи скуп ресурса имплементираних за ову апликацију и ХТТП методе које свака подржава. Комплетна имплементација вам је доступна за учење ГитХу спремиште за ову апликацију.

Та ела 14-3. Фласки АПИ ресурси

УРЛ ресурса	Опис методе	
/усер/<инт:ид>	добити	Вратите корисника.
/усер/<инт:ид>/постс/	добити	Врати све постове на логу које је написао корисник.
/усер/<инт:ид>/тимелине/ ГЕТ		Врати све постове на логу које прати корисник.
/постс/ /	добити	Врати све постове на логу.
постс/		ПОСТ Креирајте нови лог пост.
/постс/<инт:ид> /	добити	Врати пост на логу.
постс/<инт:ид>	ставити	Измените пост на логу.
/постс/<инт:ид>/цомментс/ ГЕТ		Вратите коментаре на пост на логу.
/постс/<инт:ид>/цомментс/ ПОСТ	додајте	Додајте коментар на пост на логу.
/коментари/	добити	Врати све коментаре.

УРЛ ресурса	Опис методе
/комментс/инт:ид>	доброти Врати коментар.

Имајте на уму да ресурси који су имплементирани нуде само подскуп функционалности које су доступне преко вејвлана апликације. Листа подржаних ресурса се може проширити ако је потребно, као што је откривање пратилаца, омогућавање модерирања коментара и имплементација или које друге функције које и могле затрејати АПИ клијенту.

Пагинација великих колекција ресурса ГЕТ захтеви

који враћају колекцију ресурса могујити изузетно скрути и тешки за управљање за веома велике колекције. Као и вејвлана апликације, вејвлана сервиси могу изајрати да пагинирају колекције.

Пример 14-21 показује могућу примену пагинације за листу постова на логу.

Пример 14-21. ап/апи/постс.пи: Пагинација поста

```
@апи.роуте('/постс/') деф
гет_постс(): паге =
    рекуест.аргс.гет('паге', 1, типе=инт) пагинатион =
        Пост.куери.пагинате( паге, пер_паге
            =цуррент_апп.цонфиг['ФЛАСКИ_ПОСТС_ПЕР_ПАГЕ'], еррор_аут=Фалс)
            постови = пагинатион.итемс прев = Ништа ако пагинатион.хас_прев:
                прев = урл_фор('апи.гет_постс', паге=паге-1)
                следећи = Ништа
                ако пагинација.хас_нект:
                    нект = урл_фор('апи.гет_постс', паге=паге+1) ретурн
                    јсонифи({ 'постс': [пост.то_јсон() за пост у постовима],
                        'прев_урл': прев, 'нект_урл': нект , 'коунт':
                            пагинација.укупно
                    })

```

Поље постова у ЈСОН одговору садржи ставке података као и раније, али сада је то само страница, а не комплетан сет. Ставке прев\_урл и нект\_урл садрже УРЛ-ове ресурса за претходну и наредне странице или Ништа када страница у том правцу није доступна. Вредност ројања је укупан рој ставки у колекцији.

Ова техника се може применити на све руте које враћају колекције.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецоут 14а да исте проверили ову верзију апликације. Да исте или сигурни да имате инсталiranе све зависности, такође покрените pip инсталл -р рекурементс/дев.ткт.

## Тестирање ве услуга са ХТТПије-ом

За тестирање ве услуге мора се користити ХТТП клијент. Два најчешће коришћена клијента за тестирање Питхон ве сервиса из командне линије су цУРЛ и ХТТПије. Иако су о а корисна алата, ова друга има много сажетију и читљивију синтаксу командне линије која је посе но прилагођена АПИ захтевима. ХТТПије је инсталiran са pip-ом:

```
(венв) $ pip инсталл хттпие
```

Под претпоставком да развојни сервер ради на подразумеваној адреси `хттп://127.0.0.1:5000`, ГЕТ захтев се може издати из другог прозора терминала на следећи начин:

```
(венв) $ хттп --јсон --аутх <е-пошта>:<пассворд> ГЕТ \> хттп://
127.0.0.1:5000/апи/в1/постс ХТТП/1.0 200 ОК Цонтент-Ленгтх:
7018 Цонтент-Типе: аплицијацијон/јсон Датум: нед, 22.
децем ар 2013. 08:11:24 ГМТ Сервер: Веркзеуг/0.9.4 Питхон/
2.7.3
```

```
{
    "постови": [
        ...
    ],
    "прев_урл": нулл
    "нект_урл": "хттп://127.0.0.1:5000/апи/в1/постс/?паге=2", "цоунт":
    150
}
```

О ратите пажњу на везе за пагинацију укључене у одговор. Пошто је ово прва страница, претходна страница није дефинисана, али је враћен УРЛ за до ијање следеће странице и укупан рој.

Следећа команда шаље ПОСТ захтев за додавање новог лог поста:

```
(венв) $ хттп --аутх <е-пошта>:<лозинка> --јсон ПОСТ \> хттп://
127.0.0.1:5000/апи/в1/постс / \> " оди=додајем пост из
*командна линија* "
ХТТП/1.0 201 ЦРЕАТЕД
Цонтент-Ленгтх: 360
Цонтент-Типе: аплицијацијон/јсон
Датум: нед, 22. децем ар 2013. 08:30:27
ГМТ Локација: хттп://127.0.0.1:5000/апи/в1/постс/111
Сервер : Веркзеуг/0.9.4 Питхон/2.7.3
```

```
{
```

```
"автор": "хттп://127.0.0.1:5000/апи/в1/усерс/1", " оди":  
"Додајем пост из *командне линије*.", " оди_хтмл": "<  
п>Додајем пост из <ем>командне линије</ем>.</п>", "коментари": "хттп://127.0.0.1:5000/  
апи/в1/постс/111/комментс ", "цоммент_цуонт": 0, "тиместамп": "Нед, 22. децем ар  
2013. 08:30:27 ГМТ", "урл": "хттп://127.0.0.1:5000/апи/в1/постс/111"
```

{}

Да исте користили токене за аутентификацију уместо корисничког имена и лозинке, прво се шаље ПОСТ захтев за /апи/в1/токенс/:

```
(венв) $ хттп --аутх <е-пошта>:<парола> -јсон ПОСТ \ > хттп: //  
127.0.0.1 :5000/апи/в1/токенс / ХТТП/1.0 200 ОК Цонтент-Ленгтх:  
162 Цонтент-Типе : аплицијашон/јсон Датум: Су , 04 Јан 2014  
08:38:47 ГМТ Сервер: Веркзеуг/0.9.4 Питхон/3.3.3
```

{

```
"екпиратион": 3600,  
"токен": "еијпИКСКиОјЕзОдг4МјК3МјцсИмв4цЦИ6МТМ4ОдгиОДМиНивиИВкнИјоиСФМи..."  
}
```

И сада се враћени токен може користити за упућивање позива АПИ-ју наредних сат времена тако што ћете га проследити у поље корисничког имена и оставити лозинку празну:

```
(венв) $ хттп -јсон --аутх еијпИКСК...: ГЕТ хттп://127.0.0.1:5000/апи/в1/постс/
```

Када токен истекне, захтеви ће ити враћени са грешком кода 401, што указује да је потребан нови токен.

Честитам! Ово поглавље завршава ИИ део, а тиме је завршена фаза развоја карактеристика Фласкија. Следећи корак је очигледно његово постављање, а то доноси нови скуп изазова који су предмет ИИИ дела.

ДЕО ИИИ

---

Последња миља

## ГЛАВА 15

### Тестирање

Постоје два врло до ра разлога за писање јединичних тестова. Приликом имплементације нове функционалности, јединични тестови се користе да и се потврдило да нови код ради на очекивани начин. Исти резултат се може до ити и ручним тестирањем, али, наравно, аутоматизовани тестови штеде време и труд јер се лако могу поновити.

Други, важнији разлог је тај што сваки пут када се апликација модификује, сви тестови јединица изграђени око ње могу да се изврше како и се осигурало да нема регресије у постојећем коду; другим речима, да нове промене нису утицале на начин рада старијег кода.

Јединични тестови су део Фласки-ја од самог почетка, са тестовима дизајнираним да веж ају специфичне карактеристике апликације имплементиране у класама модела азе података. Ове класе је лако тестирати ван контекста покренуте апликације, па с о зиром на то да је потребно мало труда, имплементација јединичних тестова за све карактеристике које постоје у моделима азе података је најолни начин да се оз еди да се арем тај део апликације покрене по устан и такав остаје.

Ово поглавље разматра начине за по ољшање и проширење тестирања јединица на друге о ласти апликације.

#### До ијање извештаја о покривености кода

Имати тест пакет је важан, али је подједнако важно знати колико је доар или лош. Алати за покривање кода мере колико се апликација користи јединичним тестовима и могу пружити детаљан извештај који указује на то који делови кода апликације се не тестирају. Ове информације су од непроцењиве вредности, јер се могу користити за усмеравање напора писања нових тестова у оластима којима су најпотребније.

Питхон има одличан алат за покривање кода који се прикладно назива покривеност. Можете га инсталаријати помоћу пип-а:

```
(venv) $ pip покривеност инсталације
```

Овај алат долази као скрипта командне линије која може да покрене ило коју Питхон апликацију са омогућеном покривеношћу кода, али такође пружа погоднији приступ скриптама за програмско покретање механизма покривености. Да и метрика покривености ила лепо интегрисана у команду флак тест додата у [Поглављу 7](#), може се додати опција --цовераге . Имплементација ове опције је приказана у [примеру 15-1](#).

Пример 15-1. фласки.пи: метрика покривености

```
импорт ос
импорт сис
импорт клик

ЦОВ = Нема
иФ ос.енviron.гет('ФЛАСК_ЦОВЕРАГЕ'):
    увоз покривености
    ЦОВ = цовераге.цовераге(  ранч=Труе, инклуде='апп/*')
    ЦОВ.старт()

# ...

@app.цли.цомманд()
@клицк.оптион('--цовераге/--но-цовераге', дефолт=Фалсе, хелп='Покрени
тестове под покривеношћу кода.') деф тест(цовераге):

    """"Покрени тестове јединице.""""
    ако покривеност а не ос.енviron.гет('ФЛАСК_ЦОВЕРАГЕ'):
        ос.енviron['ФЛАСК_ЦОВЕРАГЕ'] = '1' ос.екеџп(сис.екеџута  ле,
        [сис.екеџута  ле] + сис.аргв ) импорт униттест тестс =
        униттест.ТестЛоадер().дисковер('тестови')
        униттест.ТекстТестРуннер(вер  осити=2).рун(тестови) ако ЦОВ: ЦОВ.стоп()

ЦОВ.саве()
принт("Резиме покривености:")
ЦОВ.репорт()
    аседир = ос.патх.а  спатх(ос.патх.дирнаме(__file__)) цовдир =
    ос.патх.јоин(  аседир, 'тмп/цовераге')
    ЦОВ.хтмл_репорт(директори=цовдир)
    принт('ХТМЛ верзија: файле://%с/индекс.хтмл' % цовдир)
    ЦОВ.ерасе()
```

Подршка покривености кода је омогућена прослеђивањем опције --цовераге команди флак тест . Да исте додали Боолеан опцију у тест прилагођену команду,

цицк.оптион декоратер се користи. Кликните, а затим прослеђује вредност Булове заставе као аргумент функцији.

Али интегрисање покривености кода у фласки.пи скрипту представља мали про лем. Од када се опција --цовераге прими у тест() функцију, већ је прекасно за то омогућити метрику покривености; до тада је сав код у гло алном опсегу већ ио ек- цутед. Дакле, да и до или тачне метрике, скрипта се рекурзивно поново покреће након подешавања ФЛАСК\_ЦОВЕРАГЕ променљива окружења. У другом покретању, врх скрипте проналази да је променљива окружења подешена и укључује покривеност од почетка, чак и раније сав увоз апликација.

Функција цовераге.цовераге() покреће механизам покривености. Грана = Тачно опција омогућава анализу покривености грана, која поред праћења којих линија код извршења, проверава да ли су за сваки услов и Тачан и Нетачан случај извршили. Опција укључивања се користи за ограничавање анализе покривености на датотеке које налазе се унутар пакета апликације, што је једини код који тре а измерити. Без опције укључивања, сва проширења су инсталирана у виртуелном окружењу а код за саме тестове и ио укључен у извештаје о покривености — и то и унело много уке у извештају.

Након што су сви тестови извршени, функција тест() пише извештај у конзолу а такође записује лепши ХТМЛ извештај на диск. ХТМЛ верзија приказује сав извор код означен ојама које означавају линије које су покривене тестовима и оне које нису.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, ви може да покрене гит џхеџкоут 15а да провери ову верзију апликације. ција. Да исте или сигурни да имате инсталиране све зависности покрените пип инсталл -р рекуирментс/дев.ткт.

Следи пример текстуалног извештаја:

```
(венв) $ фласк тест --покривеност
...
-----
О ављена 23 теста за 6.337 с
```

У реду

Резиме покривености:

Име	Стмтс Мисс Бранцх БрПарт Цовер
-----	--------------------------------

апп/_инит_.пи апп/	32	0	0	100%
апи_в1/_инит_.пи апп/апи_в1/	3	0	0	100%
аутхентицијацијон.пи апп/апи_в1/	29	18	10	0 28%
комментс.пи апп/апи_в1/децораторс.пи	40	30	12	0 19%
	11	3	2	0 62%

апп/апи_в1/еррорс.ни апп/	17	10	0	0	41%
апи_в1/постс.ни апп/апи_в1/	36	24	8	0	27%
усерс.ни апп/аутх/_инит_.ни	30	24	12	0	14%
апп/аутх/формс.ни апп/аутх/		0	0	0	100%
виеvс.ни апп/декораторс.ни	3 45	8	8	0	70%
апп/е-пошта .ни апп/	116	91	42	0	16%
екцептационс.ни апп/майн/	14	3	2	0	69%
_инит_.ни апп/майн/	15	9	0	0	40%
еррорс.ни апп/майн/формс.ни	2		0	0	100%
апп/майн/виеvс.ни апп/	6	0 1	0	0	83%
моделс.ни	20	15	6	0	19%
	39	7	6	0	71%
	178	140	34	0	18%
	236	42	42	6	79%
-----					
ТОТАЛ	872	425	184	6	45%

ХТМЛ верзија: филе://хоме/флакс/флакси/тмп/цовераге/индек.хтмл

Извештај показује укупну покривеност од 45%, што није страшно, али није довољно. Класе модела, којима је до сада посвећена сва пажња на тестирању јединица, чине укупно 236 изјава, од којих је 79% покривено тестовима. Очигледно је виеvс.ни датотеке у главном и аутх нацрту и руте у апи\_в1 нацрту сви имају веома ниску покривеност, пошто се они не примењују ни у једној од постојећих јединица тестови. И наравно, ове метрике покривености не указују на то колико нема грешака код постоји у пројекту, пошто други фактори (као што је квалитет тестова) играју велику улогу у томе.

Наоружани овим извештајем, лако је одредити где треће да додати тестове тест пакет за пољашање покривености—али нажалост, не могу сви делови апликације тестирати лако као и модели азете података. У следећа два одељка више се говори напредне стратегије тестирања које се могу применити за преглед функција, око разазнаја и темплоче.

## Флакс тест клијент

Неки делови кода апликације се у великој мери ослањају на окружење које се креира помоћу покренуте апликације. На пример, не можете једноставно да позовете код у приказу функцију да је тестира, пошто функција можда треба да приступи променљивим контекстом Флакс-а као што су као захтев или сесија, можда очекује податке из обрасца дате у ПОСТ захтеву, и такође може захтевати пријављеног корисника. Укратко, функције прегледа могу да раде само унутар контекст захтева и покренуте апликације.

Флакс долази опремљен тестним клијентом који покушава да реши овај проблем, а време некима око им. Тест клијент реплицира окружење које постоји када је апликација ради унутар веб сервера, омогућавајући тестовима да делују као клијенти и шаљу захтеве.

Функције приказа не виде велике разлике када се извршавају под тестом клијент; захтеви се примају и усмеравају на одговарајуће функције прегледа, из којих

одговори се генеришу и враћају. Након што се функција приказа изврши, њен одговор се прослеђује тесту, који може да провери да ли је тачност.

Тестирање ве апликација

**Пример 15-2** показује оквир за тестирање јединица који користи тест клијент.

Пример 15-2. тестс/тест\_клиент.пи: оквир за тестове који користе Флакс тест клијент

```
импорт unittest
из апликације импорт цреате_апп,
д из апп.моделс импорт Усер, Роле
```

```
цласс ФлаксЦлиентТестЦасе(unittest.ТестЦасе):
    деф сетУп(селф):
        селф.апп = цреате_апп('тестињг')
        селф.апп_контект = селф.апп.апп_контект()
        селф.апп_контект.пусх() д .цреате_алл()
```

```
Роле.инсерт_ролес()
селф.клиент = селф.апп.тест_клиент(усе_коокиес=True)
```

```
деф tearDown(селф):
    д .сессион.ремове()
    д .дроп_алл()
    селф.апп_контект.поп()
```

```
деф тест_хоме_паге(селф):
    одговор = селф.клиент.гет('/')
    селф.асерптЕкуал(респонсе.статус_коде, 200)
    селф.асерптТруе('Странгер' ин респонсе.гет_дата(ас_текст=True))
```

У поређењу са тестс/тест\_ асицс.пи, овај модул додаје променљиву селф.клиент инстанце, која је Флакс тест клијентски објекат. Овај објекат излаже методе које издају захтеве апликацији. Када је тестни клијент креiran са омогућеном опцијом усе\_коокиес, он ће прихватити и послати колачиће на исти начин на који то раде претраживачи, тако да се може користити функционалност која се ослања на колачиће да опозвала контекст изменеју захтева. Конкретно, овај приступ омогућава коришћење корисничких сесија, које се чувају у колачићима.

Тест\_хоме\_паге () је једноставан пример онога што тест клијент може да уради. У овом примеру се издаје захтев за основни УРЛ апликације. Повратна вредност методе гет() тест клијента је објекат одговора Флакс који садржи одговор који је вратила позвана функција приказа. Да се проверило да ли је тест успешан, проверава се статусни код одговора, а затим се у телу одговора, до ијеном од респонсе.гет\_дата(), тражи реч „Странгер“, која је део „Здраво, Странац!“ поздрав приказан анонимним корисницима. Имајте на уму да гет\_дата()

подразумевано враћа тело одговора као низ ајтова; пассинг ac\_текст=Труе га конвертује у стринг, са којим је лакше радити.

Тест клијент такође може да шаље ПОСТ захтеве који укључују податке о расца користећи пост() метод, али подношење о разаца представља малу компликацију. Као што је о јашњено у [Поглављу 4](#), сви о расци које генерише Флакс-ВТФ имају скривено поље са ЦСРФ токеном који трећи да се достави заједно са формом. Да и могао да пошаље ЦСРФ токен, тест и морао да затражи страницу која приказује о разац, затим да рашчлани ХТМЛ који је враћен у том одговору и издвоји токен, тако да може да га пошаље са подацима о расца.

Да исте из егли про леме са ављењем ЦСРФ токенима у тестовима, оље је да онемогућите ЦСРФ заштиту у конфигурацији за тестирање. Ово је приказано у [примеру 15-3](#).

**Пример 15-3.** цон г.пи: онемогућавање ЦСРФ заштите у конфигурацији тестирања

класа ТестингЦонфиг(Цонфиг):

```
#...
ВТФ_ЦСРФ_ЕНАБЛЕД = Нетачно
```

**Пример 15-4** показује напреднији јединични тест који симулира да нови корисник региструје налог, пријављује се, потврђује налог помоћу токена за потврду и коначно се одјављује.

**Пример 15-4.** тестс/тест\_клиент.пи: симулација новог радног тока корисника са Флакс тест клијентом

класа ФлаксЦлиентТестЦасе(униттест.ТестЦасе):

```
...
# деф тест_регистер_анд_логин(селф):
    # региструјте нови одговор налога
    = селф.цилиент.пост('аутх/регистер', data={
        'е-майл': 'joxn@екампле.цом',
        'корисничко име': 'joxn',
        'парсворд': 'мачка', 'парсворд2':
        'мачка'
    })
    селф.ассерптЕкуал(респонсе.статус_цоде, 302)

    # пријавите се са новим одговором
    налога = селф.цилиент.пост('аутх/логин', data={'емайл':
        'joxn@екампле.цом', 'парсворд': 'мачка'},
        фоллов_редиреџтс=Тачно)
    селф.ассерптЕкуал(респонсе.статус_цоде, 200)
    селф.ассерптТруе(ре.сеарџх('Здраво, с+joxn!', респонсе.гет_дата(ac_текст=Труе)))

    селф.ассерптТруе('Још
    нисте потврдили свој налог' у респонсе.гет_дата(
        ac_текст=Тачно))
```

```

# пошаљи токен за потврду усер
= Усер.куери.филтер_ и(емайл='joxn@екампле.цом').фирст() токен =
усер.генерате_цонфирматион_токен() респонсе = селф.цилиент.гет('/аутх/
цонфирм/{}.формат(токен),
    фоллов_редиреџтс=Тачно)
усер.цонфирм(токен)
селф.ассертЕкуал(респонсе.статус_цоде, 200)
селф.ассертТруе('Потврдили сте свој налог' у
респонсе.гет_дата( ас_тект=Труе))

# лог оут
респонсе = селф.цилиент.гет('/аутх/логоут', фоллов_редиреџтс=Труе)
селф.ассертЕкуал(респонсе.статус_цоде, 200) селф.ассертТруе('Одјављени сте'
у респонсе.гет_дата(
    ас_тект=Тачно))

```

Тест почиње слањем о расца на путу регистрације. Аргумент података за пост() је речник са пољима о расца, која морају тачно да одговарају називима поља дефинисаним у ХТМЛ о расцу. Пошто је ЦСРФ заштита сада онемогућена у конфигурацији за тестирање, нема потребе да шаљете ЦСРФ токен са о расцем.

/аутх/регистер рута може одговорити на два начина. Ако су подаци о регистрацији исправни, преусмеравање шаље корисника на страницу за пријаву. У случају неважеће регистрације, одговор поново приказује страницу са формуларом за регистрацију, укључујући све одговарајуће поруке о грешци. Да и потврдио да је регистрација прихваћена, тест проверава да ли је статусни код одговора 302, што је код за преусмеравање.

Други део теста шаље захтев за пријаву у апликацију користећи имејл и лозинку које сте управо регистровали. Ово се ради са ПОСТ захтевом на /аутх/логин руту. Овај пут је аргумент фоллов\_редиреџтс=Труе укључен у позив пост() да и тест клијент радио као претраживач и атоматски издао ГЕТ захтев за преусмерени УРЛ. Са овом опцијом, статусни код 302 неће ити враћен; уместо тога, враћа се одговор са преусмерене УРЛ адресе.

Успешан одговор на пријаву сада и имао страницу која поздравља корисника његовим корисничким именом, а затим указује да налог тре да уде потврђен да и до и приступ. Две изјаве потврђују да је ово враћена страница. Овде је занимљиво приметити да се претрага за стрингом „Здраво, Џоне!“ не и функционисало јер је овај низ састављен од статичких и динамичких делова, па з ог начина на који је ша лон Јиња2 креiran, коначни ХТМЛ има додатни размак између ове две речи. Да и се из егла грешка у овом тесту з ог размака, користи се регуларни израз.

Следећи корак је потврда налога, што представља још једну малу препреку. УРЛ за потврду се шаље кориснику путем е-поште током регистрације, тако да не постоји једноставан начин да му се приступи са теста. Решење представљено у тесту зао илази токен

који је генерисан као део регистрације и генерише још један директно из инстанце корисника . Друга могућност и ила да се токен издвоји рашчлањивањем тела е-поште, које Флакс-Майл чува када ради у конфигурацији за тестирање.

Са токеном при руци, следећи корак теста је симулација корисника који кликне на УРЛ токена за потврду примљен путем е-поште. Ово се постиже слањем ГЕТ захтева на УРЛ за потврду, који укључује токен. Одговор на овај захтев је преусмеравање на почетну страницу, али је још једном наведено фоллов\_редиректс=Труе , тако да тест клијент автоматски захтева преусмерену страницу и враћа је. У одговору се проверава поздрав и трепћућа порука која о авештава корисника да је потврда успешна.

Последњи корак у овом тесту је слање ГЕТ захтева на руту за одјаву; да и потврдио да је ово функционисало, тест тражи трепћућу поруку у одговору.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џеџект 15 да исте проверили ову верзију апликације.

#### Тестирање ве сервиса

Флакс тест клијент се такође може користити за тестирање РЕСТфул ве услуга. [Пример 15-5](#) приказује пример класе јединичног теста са два теста.

Пример 15-5. тестс/тест\_апи.пи: РЕСТфул АПИ тестирање са Флакс тест клијентом класа АПИТестЦасе(униттест.ТестЦасе):

```
...
# деф гет_апи_хеадерс(селф, корисничко име,
    лозинка): ретурн {
    'Овлашћење': +
        'Основни' 64енцоде(
            (корисничко име + ':' + лозинка).енцоде('утф-8')).децоде('утф-8'),
        'Прихватам' 'апликацијон/јсон', 'Центент-Типе': 'апликацијон/јсон'
    }

деф тест_но_аутх(селф):
    одговор = селф.клиент.гет(урл_фор('апи.гет_постс'),
        центент_типе='апликацијон/јсон')
    селф.асерктЕквал(респонсе.статус_цоде, 401)

деф тест_постс(селф): #
    додај корисника р
    = Роле.куери.фильтер_ и(наме='Усер').фирст()
    селф.асерктИсНотНоне(р)
```

```
y = Корисник(емаил='јохн@екампле.цом', лозинка='мачка', потврђено=Тачно,
улога=r) д .сессион.адд(y) д .сессион.цоммит()
```

```
# написати
одговор на пост =
    селф.клиент.пост( '/апи/в1/
    постс', хеадерс=селф.гет_апи_хеадерс('јохн@екампле.цом', 'чат'),
    дата=јсон.думпс({' оди ': 'тело * лог* поста'}))
селф.ассертЕкуал(респонсе.статус_цоде, 201) урл =
респонсе.хеадерс.гет('Лоцацијон') селф.ассертИсНотНоне(урл)

# до иј одговор на
нови пост = селф.клиент.гет(урл,
    хеадерс=селф.гет_апи_хеадерс('јохн@екампле.цом', 'мачка'))
селф.ассертЕкуал(респонсе.статус_цоде, 200) јсон_респонсе =
јсон.лоадс( респонсе.гет_дата(ас_текст=True)) селф.ассертЕкуал('хттп://
лоцалхост' + јсон_респонсе['урл'], урл) селф.ассертЕкуал(јсон_респонсе[' оди'],
'тело * лог* поста') селф.ассертЕкуал(јсон_респонсе[' оди_хтмл'],
'<п>тело <ем> лог</ем> поста</п>')
```

Методе сетУп() и теарДовн() за тестирање АПИ-ја су исте као и за редовну апликацију, али подршка за колачиће не мора да се конфигурише јер је АПИ не користи. Метод гет\_апи\_хеадерс() је помоћни метод који враћа још и чајена заглавља која се морају послати са већином АПИ захтева. Ово укључује акредитиве за аутентификацију и заглавља која се односе на МИМЕ тип.

Тест\_no\_аутх() тест је једноставан тест који овде еђује да захтев који не укључује акредитиве за потврду идентитета уде од ијен са кодом грешке 401. Тест\_постс() тест додаје корисника у азуз података, а затим користи РЕСТфул АПИ да у аци пост на логу и онда га прочитај назад. Сви захтеви који шаљу податке у телу морају их кодирати помоћу јсон.думпс(), јер Флакс тест клијент не кодира аутоматски у ЈСОН. Слично томе, тела одговора се такође враћају у ЈСОН формату и морају се декодирати помоћу јсон.лоадс() пре него што се могу прегледати.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џеџект 15ц да исте проверили ову верзију апликације.

## Тестирање од краја до краја са селеном

Флакс тест клијент не може у потпуности да емулира окружење покренуте апликације.

На пример, или која апликација која се ослања на ЈаваСкрипт код који се покреће у клијентском претраживачу неће радити, јер ће ЈаваСкрипт код укључен у одговоре или враћен у тест ез извршења.

Када тестови захтевају комплетно окружење, нема другог из ора осим да користите прави ве претраживач повезан са апликацијом која ради на правом ве серверу. На срећу, већина ве претраживача може или атоматизована. [Селен](#) је алатка за атоматизацију ве претраживача која подржава најпопуларније ве претраживаче у три главна оперативна система. темс.

Питхон интерфејс за Селен је инсталiran са пип:

```
(венв) $ pip инсталл селениум
```

Селен захтева да се управљачки програм за жељени ве претраживач инсталира одвојено, поред самог претраживача. Постоје драјвери за све главне ве претраживаче, тако да апликација може да постави софистицирани оквир за тестирање неколико претраживача. За ову апликацију, међутим, само ће се ве претраживач Гоогле Цхроме користити за атоматизоване тестове, са одговарајућим драјвером, Цхромедривер. Ако користите мацОС рачунар са инсталационим програмом рев пакета, можете да инсталирате Цхромедривер на следећи начин:

```
(венв) $ рев инсталл цхромедривер
```

За Линук, Мицрософт Виндовс или мацОС рачунар ез рев-а, можете да преузмете о ичан програм за инсталацију [Цхромедривер-а са ве локације Цхромедривер](#).

Тестирање са Селеном захтева да апликација ради унутар ве сервера који ослушкује стварне ХТТП захтеве. Метода која ће или приказана у овом одељку покреће апликацију са развојним сервером у позадини док се тестови изводе на главној нити. Под контролом тестова, Селениум покреће ве претраживач и повезује га са апликацијом да и извршио потре не операције.

Про лем са овим приступом је у томе што након завршетка свих тестова, Флакс сервер мора или заустављен, идеално на грациозан начин, тако да позадински задаци, као што је механизам за покривање кода, могу чисто да заврше свој посао. Веркзеуг ве сервер има опцију искључивања, али пошто сервер ради изолован у сопственој нити, једини начин да затражите од сервера да се искључи је слање редовног ХТТП захтева.

[Пример 15-6](#) показује имплементацију руте за гашење сервера.

Пример 15-6. \_апп/маин/виевс.пи: пута гашења сервера

```
@маин.роуте('/схуддовн')
деф сервер_схуддовн(): ако
    није цуррент.апп.тестинг:
        а орт(404) схуддовн =
            рекуест.енвирон.гет('веркзеуг.сервер.схуддовн') ако није схуддовн:
                а орт(500) схуддовн () ретурн 'Искључивање..!'
```

Рута за искључивање ће радити само када је апликација покренута у режиму тестирања; ако га позовете у другим конфигурацијама, вратиће се одговор 404 статусног кода. Стварна процедура искључивања укључује позивање функције искључивања коју Веркзеуг излаже у окружењу. Након што позове ову функцију и врати се из захтева, развојни ве сервер ће знати да тре а да изађе на грациозан начин.

**Пример 15-7** показује изглед тест случаја који је конфигурисан да покреће тестове са Селенијумом.

Пример 15-7. тестс/тест\_селениум.пи: оквир за тестове који користе Селениум

из ве драјвера за увоз селена

```
класа СелениумТестЦасе(униттест.ТестЦасе):
    клијент = Ниједан
```

```
@классметод
деф сетУпЦласс(цлс): #
    старт Цхроме
    оптионс = ве дривер.ЦхромеОптионс()
    оптионс.адд_аргумент('хеадлесс') три:
        цлс.клиент =
            ве дривер.Цхроме(цхроме_оптионс=оптионс)
    осим:
        проћи

# прескочите ове тестове ако прегледач није могао да се покрене
ако цлс.клиент: # направите апликацију цлс.апп =
    цреате_апп('тестињг') цлс.апп_цонтект = цлс.апп.апп_цонтект()
    цлс.апп_цонтект.пуш()
```

```
# потисне евидентирање да и јединични тест излаз
остао чистим импорт логгинг логгер =
    логгинг.гетЛоггер('веркзеуг') логгер.сетЛевел("ЕРРОР")
```

```

# креирајте азу података и попуните лажним подацима
д .цреате_алл()
Роле.инсерт_ролес()
факе.усерс(10)
факе.постс(10)

# додајте администраторског корисника
админ_роле = Роле.куери.фильтер_ и(пермисионс=0кфф).фирист() админ
= Усер(емаил='joхн@екампле.цом', усернаме='joхн', пассворд ='цат',
роле=админ_роле, потврђено=Тачно)
д .сессион.адд(админ) д .сессион.цоммит()

# покрећемо Флакс сервер у нити
цлс.сервер_тхреад =
    тхреадинг.Тхреад( таргет=цлс.апп.рун, кваргс={'де уг':
        'фалсе', 'уце_релоадер':
        'Фалсе, 'уце_де угтер': 'Фалсе'})
    цлс.сервер_тхреад.старт()

@цласметход
деф теарДовнЦласс(цлс): иф
    цлс.клиент: # заустави
        Флакс сервер и претраживач цлс.клиент.гет('http://
        лоцалхост:5000/схутдовн') цлс.клиент.куит()
        цлс.сервер_тхреад.придружити()

# уништи азу
података д .дроп_алл()
д .сессион.ремове()

# уклони контекст апликације
цлс.апп_цонтект.поп()

деф сетУп(селф):
    ако није селф.клиент:
        селф.скипТест('Ве претраживач није доступан')

деф теарДовн(селф):
    проћи

```

Методе класе сетУпЦласс() и теарДовнЦласс() се позивају пре и после извршења тестова у овој класи. Подешавање укључује покретање инстанце Цхроме-а преко Селениум- овог АПИ-ја за ве драјвер , и креирање апликације и азе података са неким почетним лажним подацима за тестове. Апликација се покреће у нити помоћу методе апп.рун() . На крају апликација до ија захтев за /схутдовн, што доводи до прекида позадинске нити. Претраживач се затим затвара и тестна аза података се уклања.



Пре него што је уведен Флакс интерфејс командне линије заснован на Цлику, морали сте да покренете Флакс развојни ве сервер позивањем апп.рун() из главне скрипте апликације или користите екstenзију треће стране као што је Флакс.-Скрипта. Док је коришћење апп.рун() за покретање сервера сада замењено командом флакс рун , метода апп.рун() и даље је подржана, а овде можете видети како и даље може или корисна за сложене ситуације тестирања јединица.



Селен подржава многе друге ве претраживаче осим Цхроме-а.  
Консултујте [документацију Селена](#) ако желите да користите други ве претраживач или тестирате додатне претраживаче.

Метод сетУп() који се покреће пре сваког теста прескаче тестове ако Селениум не може да покрене ве прегледач у методи стартУпЦласс(). У [примеру 15-8](#) можете видети пример теста направљеног са Селеном.

Пример 15-8. тестс/тест\_селениум.пи: пример јединичног теста селена

класа СелениумТестЦасе(униттест.ТестЦасе):

# ...

```
деф тест_админ_хоме_паге(селф): #
    идите на почетну страницу
    селф.клиент.гет('http://лоцалхост:5000/')
    селф.асерптРтве(ре.сеарч('Здраво,\c+Странгер!','
        селф.клиент.паге_соурце))

    # идите на страницу за
    пријаву селф.клиент.финд_елемент_
        и_линк_текст('Пријава
    ').клицк() селф.асерптИн('<x1>Пријава</x1>', селф.клиент.паге_соурце)

    #
    пријавите се селф.клиент.финд_елемент_
        и_наме('емаил').
        \сенд_кеис('john@екампле.ком')
    селф.клиент.финд_елемент_
        и_наме('парола').сенд_кеис('цат')
    селф.клиент.финд_елемент_
        и_наме('су мит').клицк()
    селф.асерптРтве(ре.сеарч('Здраво,\c+john!', селф.клиент.паге_соурце))

    # идите до странице профила корисника
    селф.клиент.финд_елемент_
        и_линк_текст('Профил').клицк()
    селф.асерптИн('<x1>john</x1>', селф.клиент.паге_соурце)
```

Овај тест се пријављује у апликацију користећи администраторски налог који је креиран у сетУпЦласс() , а затим отвара страницу профила корисника. О ратите пажњу на то колико се методологија тестирања разликује од Флакс тест клијента. Приликом тестирања са Селеном, тестови шаљу команде ве претраживачу и никада не ступају у директну интеракцију са апликацијом. Тхе

команде се уско подударају са радњама које и прави корисник извршио помоћу миша или тастатуре.

Тест почиње позивом `get()` са почетне стране апликације. У претраживачу, ово узрокује да се УРЛ унесе у траку за адресу. Да и се потврдио овај корак, проверава се извор странице за „Здраво, странче!“ Поздрав.

Да и отишао на страницу за пријављивање, тест тражи везу „Пријава“ користећи `финд_елемент_и_линк_тект()`, а затим позива `цлицк()` на њој да и покренуо прави клик у прегледачу. Селен пружа неколико практичних метода `финд_елемент_и...()` које могу претраживати елементе унутар ХТМЛ странице на различите начине.

Да и се пријавио у апликацију, тест лоцира поља о расца е-поште и лозинке по њиховим именима користећи `финд_елемент_и_наме()` и затим уписује текст у њих помоћу `сенд_кеис()`. О разац се шаље позивањем `цлицк()` на дугме за слање. Персонализовани поздрав се проверава да и се осигурало да је пријава или успешна и да је претраживач сада на почетној страници.

Последњи део теста лоцира везу „Профил“ на траци за навигацију и кликне на њу.

Да и се проверило да ли је страница профил а учитана, наслов са корисничким именом се претражује у извору странице.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 15д да исте проверили ову верзију апликације. Ово ажурирање садржи миграцију азе података, па не за оправите да покренете надоградњу флакс д након што проверите код. Да исте или сигурни да имате инсталиране све зависности, такође покрените пип инсталл -р рекуирментс/дев.ткт.

Када покренете јединичне тестове са командом `флакс тест`, неће ити видљиве разлике. Јединични тест `тест_админ_хоме_паге` у примеру 15-8 ће покренути инстанцу Цхроме-а ез главе и извршити све радње на њој. Ако желите да видите радње које се изводе у стварном Цхроме прозору, коментаришите линију `оптрејнрад_формент(хроме прозор)` методи `сетУпЦласс()` тако да Селенијум

Да ли је вредно тога?

До сада се можда питате да ли је тестирање помоћу Флакс тест клијента или Селена заиста вредно труда. То је исправно питање и нема једноставан одговор.

Свидело се то вама или не, ваша апликација ће ити тестирана. Ако га сами не тестирате, ваши корисници ће постати невољни тестири; они ће пронаћи грешке, а онда ћете морати да их поправите под притиском. Једноставни и фокусирани тестови попут оних који веж ају моделе азе података и друге делове апликације који се могу извршити ван-

стране контекста апликације увек трећа спровести, јер имају веома ниску цену и око њихују правилно функционисање основних делова логике апликације.

Енд-то-енд тестови типа који Флакс тест клијент и Селенијум могу да спроведу су понекад неопходни, али због повећане сложености њиховог писања, треба да их користити само за функционалност која се не може тестирати изоловано. Апликациони код треба да буде организован тако да је могуће угурати пословну логику у модуле апликације који су независни од контекста апликације, и на тај начин се могу лакше тестирати. Код који постоји у функцијама приказа треба да буде једноставан и да делује само као танак слој који прихвата захтеве и позива одговарајуће акције у другим класама или функцијама које инкапсулирају логику апликације.

Дакле, да, тестирање је апсолутно вредно тога. Али важно је дизајнирати ефикасну стратегију тестирања и написати код који то може искористити.

## ГЛАВА 16

# Перформанс

Нико не воли споре апликације. Дуга чекања да се странице учитају фрустрирају кориснике, па је важно отворити и исправити проблеме са перформансама чим се појаве. У овом поглављу разматрају се два важна аспекта перформанси веб-апликација.

### Евидентирање спорих перформанси азе података

Када перформансе апликације полако дегенеришу с временом, то је вероватно због спорих упита азе података, који се погоршавају како величина азе података расте. Оптимизација упита азе података може или једноставна као додавање више индекса или сложена као додавање кеша између апликације и азе података. Изјава о јашњења, доступна у већини језика за упите азе података, показује кораке које аза података предузима да и извршила дати упит, често откривајући неефикасност у дизајну азе података или индекса.

Али пре него што почнете да оптимизијете упите, потреће но је утврдити који су упити они које вреди оптимизовати. Током типичног захтева може или издато неколико упита азе података, тако да је често тешко идентификовати који од свих упита су спори.

Флакс-СКЛАЦХЕМИ има опцију за снимање статистике о упитима азе података издатим током захтева. У примеру 16-1 можете видети како се ова функција може користити за евидентирање упита који су спорији од конфигурисаног прага.

Пример 16-1. апп/маин/виевс.пи: извештавање о спорим упитима азе података

```
фром flask_склалцхеми импорт гет_де уг_куериес

@маин.афтер_апп_рекуест
дeф афтер_рекуест(респонс): за
    упит у гет_де уг_куериес():
        иф куери.дуратион >= цуррент_апп.цонфиг['ФЛАСКИ_СЛОВ_ДБ_КУЕРИ_ТИМЕ']:
            цуррент_апп.логгер.варнинг(
```

'Спор упит: %с\nПараметри: %с\nТрајање: %фс\nКонтекст: %с\n' % (куери.статмент,  
куери.параметерс, куери.дуратион, куери.контекст))

#### повратни одговор

Ова функционалност је повезана са руковаоцем афтер\_апп\_рекуест, који ради на сличан начин као о рађивач ефоре\_апп\_рекуест, али се позива након што се врати функција приказа која о рађује захтев. Флакс прослеђује о јекат одговора руковаоцу афтер\_апп\_рекуест у случају да га тре а изменити.

У овом случају, о рађивач афтер\_апп\_рекуест не мења одговор; он само до ија временске интервале упита које је за елекио Флакс-СКЛАлцхеми и затим елеки споре у логер апликације који Флакс поставља на апп.логгер, пре него што врати одговор, који ће затим ити послат клијенту.

Функција гет\_де уг\_куериес() враћа упите издате током захтева као листу. Информације дате за сваки упит приказане су у **та ели 16-1**.

Та ела 16-1. Статистику упита елеки Флакс-СКЛАлцхеми

Име	Опис
изјава СКЛ наред а	
параметри Параметри који се користе са СКЛ наред ом	
старт_тиме Време када је упит издат	
енд_тиме Време када је упит вратио	
дуратион Трајање упита у секундама	
контекст Стринг који означава локацију изворног кода где је упит издат	

О рађивач афтер\_апп\_рекуест шета листом и елеки све упите који су трајали дуже од прага датог у конфигурационој променљивој ФЛАСКИ\_СЛОВ\_ДБ\_КУЕРИ\_ТИМЕ.

Евидентирање се издаје на нивоу упозорења у овој апликацији, али у неким случајевима може имати смисла третирати спора упозорења азе података као грешке.

Функција гет\_де уг\_куериес() је подразумевано омогућена само у режиму за отклањање грешака. Нажалост, про леми са перформансама азе података ретко се појављују током развоја јер се користе много мање азе података. Из тог разлога је много корисније омогућити ову опцију у производњи.

**Пример 16-2** показује промене конфигурације које су неопходне да и се омогућило праћење перформанси упита азе података у производном режиму.

Пример 16-2. цон г.пи: конфигурација за споро извештавање о упитима

```
конфигурација класе :
# ...
СКЛАЛЦХЕМИ_РЕЦОРД_КУЕРИЕС = Тачно
ФЛАСКИ_СЛОВ_ДБ_КУЕРИ_ТИМЕ = 0,5 #
...
```

СКЛАЛЦХЕМИ\_РЕЦОРД\_КУЕРИЕС говори Флакс-СКЛАЛцхеми да омогући снимање статистике упита. Праг спорог упита је подешен на пола секунде. О `е` конфигурационе варија `ле` су укључене у основну класу Цонфиг, тако да `ће` ити омогућене за све конфигурације.

Кад год се открије спор упит, унос `ће` ити уписан у Флакс-ов дневник апликације. Да `исте` могли да сачувате ове уносе дневника, логер мора `ити` конфигурисан. Конфигурација евидентирања у великој мери зависи од платформе на којој се апликација налази. Неки примери су приказани у [Поглављу 17.](#)



Ако сте клонирали Гит спремиште апликације на ГитХу `-у`, можете покренути гит цхецкоут 16а да `исте` проверили ову верзију апликације.

## Изворни код Про линг

Други могући извор про `лема` са перформансама је велика потрошња процесора, узрокована функцијама које о `ављају` тешке рачунарске послове. Профилери извornог кода су корисни у проналажењу најспоријих делова апликације. Профилер прати покренуту апликацију и `ележи` функције које се позивају и колико дуго је свакој потребе `но` да се покрене. Затим производи детаљан извештај који приказује најспорије функције.



Профилисање се о `ично` ради само у развојном окружењу. Профилер извornог кода чини да апликација ради много спорије него иначе, јер мора да посматра и `ележи` све што се дешава у реалном времену. Профилисање на производном систему се не препоручује, осим ако се не користи лагани профилатор посе `но` дизајниран за рад у производном окружењу.

Флаксов развојни ве `сервер`, који долази из Веркзеуг-а, може опционо да омогући Питхон профилер за сваки захтев. [Пример 16-3](#) додаје нову опцију командне линије у апликацију која покреће ве `сервер` под профилером.

Пример 16-3. фласки.пи: покретање апликације под захтевом пролера

```
@апп.ци.цомманд()
@цицк.оптион('--ленгтх', подразумевано=25,
    хелп='Број функција које трећа укључити у извештај профилера.')
@цицк.оптион('--профиле-дир', дефаулт=Ништа,
    хелп='Директоријум где се чувају датотеке са подацима профилера.')
деф профиле(ленгтх, профиле_дир):
    """Покрените апликацију под профилером кода."""
    фром веркзеуг.центри .профилер импорт ПрофилерМидлеваре
    апп.всги_апп = ПрофилерМидлеваре(апп.всги_апп, рестрикционс=[ленгтх], профиле_дир=профиле_дир)

    апп.рун(де уг=Фалсе)
```

Ова команда прилаже ПрофилерМидлеваре из Веркзеуг-а апликацији, преко њеног атријута `всги_апп`. ВСГИ међувера се позива сваки пут када већ сервер пошаље захтев апликацији и може да измене начин на који се захтев рукује, у овом случају хватањем података о профилисању. Имајте на уму да се апликација затим програмски покреће помоћу методе `апп.рун()`.



Ако сте клонирали Гит спремиште апликације на ГитХуу, можете покренути гит џеџкоут 16 да ћисте проверили ову верзију апликације.

Када се апликација покрене са фласк профилом, конзола ће приказати статистику профилера за сваки захтев, који ће укључивати 25 најспоријих функција. Опција `--ленгтх` се може користити за промену броја функција приказаних у извештају.

Ако је дата опција `--профиле-дир`, подаци профиле за сваки захтев се чувају у датотеки у датом директоријуму. Датотеке са подацима профилера се могу користити за генерирање детаљнијих извештаја који укључују графикон позива. За више информација о Питхон профилеру, погледајте [званичну документацију](#).

Припреме за распоређивање су завршене. Следеће поглавље ће вам дати преглед шта можете да очекујете приликом постављања ваше апликације.

## ГЛАВА 17

# Деплоимент

Сервер за ве развој који долази у пакету са Фласком није довољно ро устан, ез едан или ефикасан за рад у производном окружењу. У овом поглављу се испитују опције примене у производњи за Флак апликације.

## Деплоимент Ворк вл

Без о зира на коришћени метод хостовања, постоји низ задатака који се морају о авити када је апликација инсталирана на производном серверу. То укључује креирање или ажурирање та ела азе података.

Ручно покретање ових задатака сваки пут када се апликација инсталира или надогради је склона грешкама и одузима много времена. Уместо тога, команда која о авља све потре не задатке може се додати у фласки.пи.

**Пример 17-1** показује имплементацију команде деплои која је прикладна за Фласки.

Пример 17-1. фласки.пи: команда распоређивања

```
фром flask_миграте импорт упграде фром
апп.моделс импорт Роле, Усер
```

```
@манагер.цомманд
```

```
деф деплои():
```

```
    """Покрени задатке постављања."""

```

```
    # миграра азу података на најновију верзију
    надоградње()

```

```
    # креирајте или ажурирајте корисничке
    улоге Роле.инсерт_ролес()
```

```
# уверите се да сви корисници прате сами се   е
Усер.адд_селф_фолловс()
```

Све функције које се позивају овом командом су креиране раније; они се само позивају заједно из једне команде да и се поједноставила примена апликације.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 17а да исте проверили ову верзију апликације.

Све ове функције су дизајниране на начин да не изазивају проблеме ако се изврше више пута. Дизајнирање функција ажурирања на овај начин омогућава покретање само ове команде постављања сваки пут када се изврши инсталација или надоградња је због потребе да ринете о нежељеним ефектима узрокованим функцијом која се покреће у погрешно време.

## Евидентирање грешака током производње

Када апликација ради у режиму за отклањање грешака, Веркзеугов интерактивни програм за отклањање грешака се појављује кад год дође до грешке. Траг грешке стека се приказује на већим страници и могуће је погледати изворни код и чак проценити изразе у контексту сваког оквира стека користећи Флакс интерактивни веб-визуелери де агер.

Програм за отклањање грешака је одличан алат за отклањање грешака у апликацији током развоја, али очигледно се не може користити у производњи. Грешке које се јављају у производњи се пређуткују и уместо тога корисник дође на дискретну страницу грешке код 500. Али на срећу, трагови стека ових грешака нису потпуно изгубљени, пошто их Флакс уписује у датотеку евиденције.

Током покретања, Флакс креира инстанцу Питхон-ове класе логгинг.Логгер и прилаже је инстанци апликације као ап.логгер. У режиму за отклањање грешака, овај логер пише у конзолу, али у производном режиму нема подразумевано конфигурисаних рукавалца за њега. Осим ако није додат рукавалац, евиденције се не чувају. Промене у [Примеру 17-2](#) конфигуришу рукавалац евиденције који шаље грешке које се јављају током извођења под производном конфигурацијом на адресу е-поште администратора конфигурисану у `ФЛАСКИ_АДМИН` поставци.

Пример 17-2. цон.г.пи: слање е-поште за грешке у апликацији

```
класа ПродуцционЦонфиг(Цонфиг):
    # ...
    @классметод
    деф инит_апп(цлс, апп):
        Цонфиг.инит_апп(апп)
```

```

# грешка е-поште администраторима увоз
евиденције са логинг.хандлерс импорт
СМТПХандлер акредитиви = Ниједан сигуран =
Нема ако гетаттр(цлс, 'МАИЛ_УСЕРНАМЕ', Ноне)
није Ништа:

акредитиви = (цлс.МАИЛ_УСЕРНАМЕ, цлс.МАИЛ_ПАССВОРД)
ако гетаттр(цлс, 'МАИЛ_УСЕ_ТЛС', Ништа): сецуре = ()
mail_хандлер = СМТПХандлер(
mailхост=(цлс.МАИЛ_СЕРВЕР, цлс.МАИЛ_ПОРТ),
фромаддр=цлс.ФЛАСКИ_МАИЛ_СЕНДЕР,
тоаддрс=[цлс.ФЛАСКИ_АДМИН],
су јеџт=цлс.ФЛАСКИ_МАИЛ_СУБЈЕЦТ_ПРЕФИКС „Грешка у апликацији“,
+ акредитиви=акредитиви), сецуре .
логгер.аддХандлер(mail_хандлер)

```

Подсетимо се да све конфигурационе класе имају статичку методу иницијализација() коју позива цреате\_апп(), која до сада није коришћена. У имплементацији ове методе за класу ПродуцтионЦонфиг , дневник апликације је сада конфигурисан са руководоцима евиденције који шаље грешке примаоцу е-поште.

Ниво евидентирања евиденције е-поште је подешен на логинг.ЕРРОР, тако да ће се само оз иљни про леми слати е-поштом. Поруке евидентирање на њиховима могу се евидентирати у датотеку, системски дневник или ило које друго подржано одредиште додавањем одговарајућих руковаца евиденције. Начин евидентирања који се користи за ове поруке у великој мери зависи од платформе за хостовање.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џеџкоут 17 да исте проверили ову верзију апликације.

## Цлоуд Деплоимент

Тренд у хостингу апликација је хостовање „у о лаку“, али то може значити много различитих ствари. На најосновнијем нивоу, хостинг у о лаку може значити да је апликација инсталована на једном или више виртуелних сервера, који за све намере и сврхе раде и осећају се као физичке машине, али у стварности су виртуелне машине којима управља клауд оператор. Пример ових типова сервера су они који су доступни преко услуге ЕЦ2 од Амазон Ве Сервицес (ABC). Постављање апликације на виртуелни сервер је слично традиционалном постављању на наменски сервер, као што је описано касније у овом поглављу.

Напреднији модел имплементације заснован је на контејнерима. Контејнер изолује апликацију у слици апликације и њеног окружења. Слика контејнера укључује апликацију плус све зависности које су јој потребне за покретање. Контејнерска платформа, као што је Доцкер, може затим да инсталира и изврши унапред генерисану слику контејнера на ило ком систему у којем се покреће.

Друга опција имплементације, формално позната као Платформа као услуга (ПааС), ослоњаја програмера апликација од свакодневних задатака инсталирања и одржавања хардверских и софтверских платформи на којима апликација ради. У ПааС моделу, провајдер услуга нуди потпуно управљану платформу на којој апликације могу да раде. Све што програмер апликације треба да уради је да отпреми код апликације на сервере које одржавају провајдер, након чега он автоматски постаје доступан, очично у року од неколико секунди. Већина ПааС провајдера нуди начине за динамичко „скалирање“ апликације додавањем или уклањањем сервера по потребе и како и се одржао корак са ројем примљених захтева.

Остатак овог поглавља нуди увод у Хероку (један од најпопуларнијих ПааС провајдера), Доцкер контејнере и на крају традиционалне примене, које су погодне за наменске или виртуелне сервере.

## Хероку платформа

Хероку је ио један од првих ПааС провајдера, који послује од 2007. Хероку платформа је веома флексијилна и подржава дугу листу програмских језика, укључујући Питхон. Да и применио апликацију на Хероку, програмер користи Гит да гурне апликацију на Херокуов специјални Гит сервер, који автоматски покреће инсталацију, надоградњу, конфигурацију и примену апликације.

Хероку користи рачунарске јединице које се зову динос за мерење употребе и наплату услуге. Најчешћи тип дино-а је већи дино, који представља инстанцу већег сервера. Апликација може да повећа свој капацитет за овадије захтева тако што ће применити више већих диноса, од којих сваки покреће инстанцу апликације. Други тип дино-а је раднички динамофон који се користи за оваљање позадинских послова или других задатака подршке.

Платформа пружа велики рој додатака и додатака за азе података, подршку путем е-поште и многе друге услуге. Следећи одељци проширују неке од детаља укључених у постављање Фласкија на Хероку.

### Припрема апликације

Да исте

радили са Херокуом, апликација мора ити смештена у Гит спремишту. Ако радите са апликацијом која је хостована на удаљеном Гит серверу, као што је ГитХу или Бит уцкет, клонирање апликације ће креирати локално Гит спремиште које је савршено за

користите са Херокуом. Ако апликација већ није хостована у Гит спремишту, мораћете да је креирате за њу на вашој машини за развој.



Ако планирате да хостујете своју апликацију на Хероку-у, до ра је идеја да почнете да користите Гит од самог почетка. ГитХу има водиче за инсталацију и подешавање за три главна оперативна система у свом водичу за [помоћ](#).

### Креирање Хероку налога

Морате да [направите налог са Херокуом](#) пре него што удете могли да користите услугу. Хероку пружа есплатан ниво који вам омогућава да угостите неколико једноставних апликација, тако да је ово одлична платформа за експериментисање.

### Инсталирање Хероку ЦЛИ

За рад са Хероку услугом, [Хероку ЦЛИ](#) мора ити инсталација. Ово је клијент командне линије који управља интеракцијама са услугом. Хероку ојезејује инсталаторе за три главна оперативна система.

Прва ствар коју треба да урадите након инсталација ЦЛИ је да се аутентификујете са својим Хероку налогом преко хероку команде за пријаву:

```
$ heroku login
Унесите своје Хероку акредитиве.
Емаил: <ваша-е-адреса>
Лозинка: <ваша-лозинка>
```



Важно је да ваш CCX јавни кључ буде учитан у Хероку, јер је то оно што омогућава гит пусх команду. Очино команда за пријаву аутоматски креира и отпрема CCX јавни кључ, али команда хероку кеис:адд се може користити за отпремање вашег јавног кључа одвојено од команде за пријаву или ако треба да отпремите додатне кључеве.

### Креирање апликације

Следећи корак је креирање апликације. Пре него што се ово уради, апликација мора да буде под Гит изворном контролом. Ако сте користили ГитХу спремиште да исте пратили код у овој књизи, онда већ имате Гит спремиште. Ако не, мораћете да га креирате сада. Да исте регистровали апликацију у Хероку-у, покрените следећу команду из директоријума највишег нивоа апликације:

```
$ хероку цреате <аппнаме>
Креирање <аппнаме>...
урађено http://<аппнаме>.херокуапп.цом/ | http://git.heroku.com/<име апликације>.git
```

Називи Хероку апликација морају ити јединствени за све клијенте, тако да морате да смислите име које не преузима ниједна друга апликација. Као што показује излаз команде креирања , након што се апликација примени, иће доступна на `https://<апп-наме>.herokuapp.com`. Хероку такође подржава коришћење прилагођеног имена домена за вашу апликацију.

Као део креирања апликације, Хероку креира Гит сервер посвећен вашој апликацији на `https://git.heroku.com/<аппнаме>.git`. Команда креирања додаје овај сервер у ваше локално Гит спремиште као гит даљински са именом хероку:

```
$ git remote add heroku *
      даљински хероку
      Преузми УРЛ: https://git.heroku.com/<аппнаме>.git Пусх
      УРЛ: https://git.heroku.com/<аппнаме>.git ХЕАД грана:
      (непознато)
```

Команда фласк захтева да променљива окружења ФЛАСК\_АПП уде подешена да ради. Да исте или сигурни да су све команде које се извршавају у Хероку окружењу успешне, до раје идеја да региструјете ову променљиву окружења тако да је увек подешена када Хероку извршава команде повезане са овом апликацијом. Ово се може урадити помоћу нареде конфигурације:

```
$ хероку конфиг:сет ФЛАСК_АПП=фласки.py
Подешавање ФЛАСК_АПП и поновно покретање <аппнаме>...
урађено, в4 ФЛАСК_АПП: фласки.py
```

Овај ефикаснији азимут података Хероку подржава Постгрес азимут података као додатак. Ниво есплатне услуге укључује малу азу података до 10.000 редова. Да исте приложили Постгрес азу података вашој апликацији, користите следећу команду:

```
$ хероку аддонс:цреате хероку-постгрескл:х0 и-дев
Креирање хероку-постгрескл:х0 и-дев на <аппнаме>... есплатна
азимут података је креирана и доступна је
! Ова азу података је празна. Ако надоградите, можете пренети!
подаци из друге азимут података са pg:копи Креиран постгрескл-
цу иц-41298 као ДАТАБАСЕ_УРЛ Користите хероку аддонс:доџ хероку-
постгрескл за преглед документације
```

Као што показује излаз команде, када се апликација покрене унутар Хероку платформе, видеће локацију азимут података и акредитиве у променљивој окружењу ДАТАБАСЕ\_УРЛ . Формат ове променљиве је УРЛ, тачно у формату који СКЛАДЦИ очекује. Подсетимо се да скрипта конфигурације користи вредност ДАТАБАСЕ\_УРЛ ако је дефинисана, тако да ће веза са Постгрес азимут података радити аутоматски.

## Подешавање

евиденције Евидентирање фаталних грешака путем е-поште је додато раније, али је поред тога важно конфигурисати евиденцију мањих категорија порука. До ар пример ових типова порука су упозорења за споре упите азе података додана у [поглављу 16.](#)

Хероку узима у о зир сваки излаз који је апликација написала у стдоут или стдерр евиденције, тако да је потребно додати руковалац евидентирањем да и се генерира овај излаз. Хероку снима излаз евиденције и постаје доступан преко Хероку клијента помоћу команде хероку логс .

Конфигурација евидентирања може се додати у класу ПродуцционЦонфиг у њеном статичком методу иниит\_апп() . Али пошто је ова врста евидентирања специфична за Хероку, овљи приступ је да се дефинише нова конфигурација посве но за ову платформу, остављајући ПродуцционЦонфиг као основну конфигурацију за различите типове производних платформи. Класа ХерокуЦонфиг је приказана у [примеру 17-3.](#)

Пример 17-3. ћон г.пи: Хероку кон гурација

```
цласс ХерокуЦонфиг(ПродуцционЦонфиг):
    @цласметод деф иниит_апп(цлс, апп):
        ПродуцционЦонфиг.иниит_апп(апп)
```

```
# логовање у
стдерр увоз
евиденције из евиденције импорт
СтреамХандлер файле_хандлер =
СтреамХандлер()
филе_хандлер.сетЛевел(логгинг.ИНФО) апп.логгер.аддХандлер(филе_хандлер)
```

Када Хероку изврши апликацију, мора да зна да се ова нова конфигурација мора користити. Инстанца апликације креирана у фласки.пи користи променљиву окружења ФЛАСК\_ЦОНФИГ да и знала коју конфигурацију да користи, тако да ова променљива мора да се подеси на одговарајући начин у Хероку окружењу. Променљиве окружења за Хероку окружење се постављају помоћу команде конфиг:сет клијента Хероку :

```
$ хероку цонфиг:сет ФЛАСК_ЦОНФИГ=хероку
Подешавање ФЛАСК_ЦОНФИГ и поново покретање <аппнаме>...
урађено, в4 ФЛАСК_ЦОНФИГ: хероку
```

Да исте повећали је едност ваше апликације, до је идеја да конфигуришете низ који је тешко погодити као тајни кључ апликације, који се користи за потписивање корисничке сесије и токена за аутентификацију. Основна класа Цонфиг укључује атри јут СЕЦРЕТ\_КЕИ за ову сврху и поставља своју вредност из променљиве окружења истог имена ако постоји. Када радите на апликацији у вашем развојном систему, у реду је оставити ову променљиву недефинисану и пустити класи Цонфиг да конфигурише чврсто кодиран

вредност, али на производној платформи је изузетно важно поставити јак тајни кључ који никоме није познат, пошто ће процурили кључ омогућити нападачу да фалсификује садржај корисничке сесије или генерише важеће токене. Да исте свој кључ учинили сигурним, само поставите променљиву окружења СЕЦРЕТ\_КЕИ на јединствени стринг који се никде не чува:

```
$ хероку цонфиг:сет СЕЦРЕТ_КЕИ=d68653675379485599ф7876а3 469а57
Подешавање СЕЦРЕТ_КЕИ и поновно покретање <аппнаме>... урађено, в4
СЕЦРЕТ_КЕИ: d68653675379485599ф786
```

Постоји много начина за генерисање насумичних низова који су прикладни да се користе као тајни кључеви. То можете учинити са Питхон-ом на следећи начин:

```
(венв) $ питхон -ц "увези ууид; принт(ууид.ууид4().хек)"
d68653675379485599ф7876а3 469а57
```

Подешавање е-

поште Хероку не о је еђује СМТП сервер, тако да мора да се конфигурише спољни сервер. Постоји неколико додатака независних производођача који интегришу подршку за слање е-поште спремну за производњу са Херокуом, али за потре је тестирања и евалуацијеовољно је користити подразумевану конфигурацију Гмаил-а наслеђену из основне класе Цонфиг .

Будући да може представљати је једносни ризик уграђивање акредитива за пријаву директно у скрипту, корисничко име и лозинка за приступ Гмаил СМТП серверу су дати као променљиве окружења (ако још нисте, веома је до ра идеја да уместо користећи свој лични налог е-поште креирате секундарну е-пошту коју ћете користити за тестирање):

```
$ хероку цонфиг:сет МАИЛ_УСЕРНАМЕ=<иоур-гмаил-усернаме> $
хероку цонфиг:сет МАИЛ_ПАССВОРД=<иоур-гмаил-пассворд>
```

Добавање захтева највишег нивоа ле

Хероку инсталира зависности пакета из датотеке рекуирментс.ткт ускладиштене у директоријуму највишег нивоа апликације. Све зависности у овој датотеци ће ити увезене у виртуелно окружење којим управља Хероку као део примене.

Хероку датотека са захтевима мора да садржи све уо ичайене захтеве за производну верзију апликације, плус псицопг2 пакет који омогућава СКЛ-Алхемији приступ Постгрес ази података. Датотека хероку.ткт са овим зависностима може да се дода у директоријум Захтеви, а затим да се увезе из датотеке рекуирментс.ткт највишег нивоа као што је приказано у [Примеру 17-4.](#)

Пример 17-4. Захтеви.ткт: Хероку захтеви ле

-р Захтеви/хероку.ткт

Омогућавање је едног ХТТП-а са

Флак-ССЛифи-ом Када се корисник пријави у апликацију слањем корисничког имена и лозинке у вео расцу, ове вредности су у опасности да их пресретне злонамерна трећа страна, као што је већ неколико пута дискутовано. Током развоја ово није пројем, али овај ризик треја да елиминисати када примените апликацију на производном серверу. Да је исте спречили отварање корисничких акредитива док су у транзиту, неопходно је користити је едан ХТТП, који шифрује сву комуникацију између клијената и сервера користећи криптографију јавног кључа.

Хероку чини све апликације којима се приступа на домену херокуап.цом доступним на `http://` и `https://` је потребно за конфигурацијом. Пошто апликација ради на Хероку домену, користиће Херокуов сопствени ССЛ сертификат. Једина неопходна радња за потпуну сигурност апликације је пресретање свих захтева послатих на `http://` интерфејс и њихово преусмеравање на `https://`, што је управо оно што Флак ССЛифи екstenзија ради.

Као и оично, Флак-ССЛифи се инсталира са пип:

```
(веб) $ pip install flask-sslify
```

Код који активира ову екстензију се додаје у формату функцију апликације, као што је приказано у примеру 17-5.

Пример 17-5. `app/__init__.py`: преусмеравање свих захтева каје једном ХТТП-у

```
def create_app(config_name):
    # ...
    if app.config['SSL_REDIRECT']:
        from flask_sslify import SSLify
        SSLify(app)
    # ...
```

Подршка за ССЛ треба да буде омогућена само у производном режиму и само када га платформа подржава. Да се олакшало укључивање и искључивање ССЛ-а, дodata је нова конфигурациона варијаја под називом `SSL_REDIRECT`. Основна класа Цонфиг постављаје на Фалс, тако да се ССЛ преусмеравање не користе подразумевано, а класа ХерокуЦонфиг је замењује тако да се преусмеравања издају само на тој конфигурацији. Имплементација ове конфигурационе променљиве је приказана у примеру 17-6.

Пример 17-6. цон г.пи: подешавање употребе е ССЛ-а

```
конфигурација класе :
# ...
SSL_РЕДИРЕЦТ = Нетачно

класа ХерокуЦонфиг(ПродуцтионЦонфиг):
# ...
SSL_РЕДИРЕЦТ = Тачно ако ос.енвирон.гет('ДИНО') друго Нетачно
```

Вредност SSL\_РЕДИРЕЦТ у ХерокуЦонфиг је постављена на Тачно само ако постоји променљива окружења ДИНО . Ову променљиву поставља Хероку у свом окружењу, тако да коришћење Хероку конфигурације за локално тестирање не активира SSL преусмеравања.

Са овим променама, корисници ће ити приморани да користе SSL сервер када приступају апликацији на Хероку-у—али постоји још један детаљ који трећа да се о ради да и ова подршка ила потпуна. Када се користи Хероку, клијенти се не повезују директно на апликацију, већ на реверзи прокси сервер. О рнутти прокси сервер прима захтеве од многих апликација и прослеђује их свакој од њих према потреби. У овом типу подешавања, само прокси сервер ради у SSL режиму; SSL веза се прекида на прокси серверу, а апликације примају прослеђене захтеве са прокси сервера уз шифровања. Ово представља проблем када апликација треба да генерише апсолутне УРЛ адресе, јер у Flask апликацији ојекат захтева описује прослеђени захтев, који није шифрован, а не оригинални захтев који клијент шаље преко шифроване везе.

Пример проблема који ово може да изазове је генерисање веза за потврду налога или ресетовање лозинке које се шаљу е-поштом корисницима. Када се url\_for() позове са \_эктернал=True да генерише апсолутни УРЛ за ове везе, Flask ће користити http:// за њих, јер не зна да постоји ојакутни прокси који прихвата шифроване везе извана .

Прокси сервери прослеђују информације које описују оригинални захтев од клијента преусмереним већим серверима кроз прилагођена ХТТП заглавља, тако да је могуће утврдити да ли корисник комуницира са апликацијом преко SSL-а гледајући ова заглавља. Верзје ојакуте енџије ВСГИ међуверски софтвер који проверава прилагођена заглавља са прокси сервером и у складу са тим ажурира ојекат захтева тако да, на пример, request.is\_securerodражава стање шифровања захтева који је клијент послao ојакутном прокси серверу и не захтев који је прокси сервер затим проследио апликацији. Пример 17-7 показује како додати Прокифик средњи софтвер у апликацију.

Пример 17-7. цон г.пи: додавање подршке за прокси сервере

```
класа ХерокуЦонфиг(ПродуционЦонфиг):
    ...
    # @классметход
    деф инициализација(апл, апли):
        # ...

        # о ради заглавља о рнутог прокси
        сервера из веркзеуг.цонтри .фикерс импорт
        Прокифик апли.всги_апл = Прокифик(апли.всги_апл)
```

Међувер је додат у методу иницијализације за Хероку конфигурацију.

ВСГИ међувера као што је Прокифик се додаје омотањем ВСГИ апликације.

Када дође захтев, средњи софтвер до ија прилику да прегледа окружење и изврши промене пре него што се захтев о ради. Прокифик средњи софтвер је неопходан не само за Хероку, већ и за сваку примену која користи о рнути прокси сервер.

Покретање производног ве

сервера Хероку очекује од апликација да покрену сопствени производни ве сервер и конфигуришу га да слуша захтеве на појму порта постављеном у променљивој окружењу ПОРТ.

Развојни ве сервер који долази са Флакс-ом ће се у овој ситуацији понашати веома лоше јер није дизајниран да ради у производном окружењу. Два ве сервера спремна за производњу која до по раде са Флакс апликацијама су [Гуницорн](#) и [уВСГИ](#).

До по је инсталирати ода рани ве сервер у локалном виртуелном окружењу, како и се могао тестирати на начин сличан ономе како ће радити у Хероку окружењу.

На пример, Гуницорн се инсталира на следећи начин:

```
(веб) $ пип инсталл гуницорн
```

Да исте покренули апликацију локално под Гуницорн-ом, користите следећу команду:

```
(веб) $ гуницорн фласки:апп
[2017-08-03 23:54:36 -0700] [ИНФО] Покретање гуницорн 19.7.1
[2017-08-03 23:54:36 -0700] [ИНФО] Слушање на: http://127.0.0.1:8000 (68982)
[2017-08-03 23:54:36 -0700] [ИНФО] Коришћење радника:
синхронизација [2017-08-03 23:54:36 -0700] [ИНФО] Покретање радника са пид-ом: 68985
```

Аргумент фласки:апп говори Гуницорн-у где се налази инстанца апликације.

Име дато пре дводаљаке је пакет или модул који дефинише ову инстанцу, док је име после дводаљаке стварно име инстанце апликације. Имајте на уму да Гуницорн подразумевано користи порт 8000, а не 5000 као Флакс. Као и ве сервер за развој Флакс, можете изабрati из Гуницорн-а помоћу Цтрл+Ц.



Гуницорн ве сервер не ради на Мицрософт Виндовс-у. Други препоручени ве сервер, уВСГИ, ради на Виндовс-у, али може ити тешко инсталаријати јер је написан у извornом коду. Ако желите да тестирајте примену Херокуа на вашем Виндовс систему, можете користити [Ваитресс](#), који је још један чисти Питхон ве сервер који је на много начина сличан Гуницорн-у, али има предност што у потпуности подржава Виндовс. Коно арица је инсталирана са пип:

```
(венв) $ пип инсталл коно арица
```

Да исте покренули ве сервер Ваитресс, користите команду ваитресс-серве:

```
(венв) $ ваитресс-серве --порт 8000 фласки:апп
```

#### Добавање Проц

ле Хероку-а тре да зна коју команду тре а користити за покретање апликације. Ова команда је дата у посе ној датотеци под називом Проц ле. Ова датотека мора ити укључена у директоријум највишиг нивоа апликације.

**Пример 17-8** приказује садржај ове датотеке.

#### Пример 17-8. Проц ле: Хероку Проц ле

ве :гуницорн фласки:апп

Формат за Проц ле је веома једноставан: у свакој линији се даје име задатка, праћено двотачком и затим команда која покреће задатак. Име задатка ве је посе но; Хероку га препознаје као задатак који покреће ве сервер. Хероку ће овом задатку дати променљиву окружења ПОРТ постављену на порт на коме апликација тре да слуша захтеве. Гуницорн подразумевано поштује променљиву ПОРТ ако је постављена у окружењу, тако да нема потребе да је укључујете у команду за покретање.



Ако користите Мицрософт Виндовс или вам је потреба на да ваша апликација уде у потпуности компати илна са том платформом, уместо тога можете да користите ве сервер Ваитресс:

```
ве :ваитресс-серве --порт=$ПОРТ фласки:апп
```



Апликације могу да декларишу додатне задатке са називима који нису ве у Проц ле. Сваки задатак укључен у Проц ле ће ити покренут на сопственом динамофону.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 17ц да исте проверили ову верзију апликације. Ако користите Мицрософт Виндовс, покрените гит цхецкоут 17ц вантресс да исте проверили верзију апликације која је конфигурисана да користи Ве сервер Вантресс уместо Гунизорн-а.

### Тестирање помоћу Хероку Лоцал -а

Хероку ЦЛИ укључује локалну команду, која се користи за локално покретање апликације на веома сличан начин као што ради на Хероку серверима. Међутим, променљиве окружења као што је ФЛАСК\_АПП нису доступне када се апликација покреће локално. Локална команда хероку тражи променљиве окружења које конфигуришу апликацију у датотеци под називом .енв у директоријуму највишег нивоа апликације. На пример, .енв датотека може да садржи следеће променљиве:

```
ФЛАСК_АПП=фласки.пи
ФЛАСК_ЦОНФИГ=хероку
МАИЛ_УСЕРНАМЕ=<ваше-гмаил-корисничко
име> МАИЛ_ПАССВОРД=<ваша-гмаил-лозинка>
```



Пошто .енв датотека садржи лозинке и друге осетљиве информације о налогу, никада је не и тре ало додавати у контролу извора.

Пре него што се апликација може покренути, задатак постављања тре а да се изврши да и се подесила аза података. Једнократни задаци се могу извршити командом лоцал:рун :

```
(венв) $ хероку лоцал:рун фласк деплои [ОКАИ]
Учитана ЕНВ.енв датотека као КЕИ=ВАЛУЕ Формат ИНФО Контекст
импл СКЛитеИмпл.
ИНФО Претпоставља нетрансакциони ДДЛ.
Инфо за надоградњу за рад -> 38ц4е85512а9, почетне информације о миграцији
Подаци на 38Ц4Е85512А9 -> 456А945560Ф6, пријава за подршку Акцион-а 456А945560Ф6 ->
190163627111 -> 56163627111 -> 56еđ7д33Де8д, угради у корисник Информације о коришћењу
Информације о корисник Информације о коришћењу Информације о коришћењу Радите надоградњу
д66Ф086Б258 -> 198Б0ЕЕБЦФ9, кеширање аватар Хасхес Информације о раду 198Б0ЕЕБЦФ9 ->
1Б966Е7Ф4Б9Е, Постави модела Подаци на надоградњу 1Б966Е7Ф4Б9Е -> 288Цд3Ц5А8, 656д3Ц5А8 ->
2356А38169АЕ, АДВОКАТМЕНТС ИНФО РУНГРАДИОНС -> 51Ф5ЦЦФБА190
```

Локална команда хероку чита Проц ле и извршава задатке дефинисане њиме:

```
(веб) $ хероку лоцал [OKAI]
Учитана ЕНВ .енв датотека као КЕИ=ВАЛУЕ Формат 11:37:49 AM
ве .1 | [ИНФО] Покретање гуницирн 19.7.1 11:37:49 ве .1 | [ИНФО]
Слушам на: http://0.0.0.0:5000 (91686)
11:37:49 AM ве .1 | [ИНФО] Коришћење радника: синц 11:37:49
AM ве .1 | [ИНФО] Боотинг воркер са пид-ом: 91689
```

Излаз евиденције свих задатака покренутих овом командом се консолидује у један ток који се штампа на конзоли, са сваким редом са префиксом временске ознаке и назива задатка.

Локална команда хероку такође омогућава симулацију употребе више динамометара за скалирање апликације. Следећа команда покреће три ве радника, од којих сваки слуша на другом порту:

```
(веб) $ хероку локални ве =3
```

Примена помоћу гит пусх -а

Последњи корак у процесу је отпремање апликације на Хероку сервере. Уверите се да су све промене унете у локално Гит спремиште, а затим користите гит пусх хероку мастер да отпремите апликацију на хероку даљински:

```
$ гит пусх хероку мастер Бројање
о јеката: 502, готово.
Делта компресија користећи до 8 нити.
Компресија о јеката: 100% (426/426), завршено.
Предмети за писање: 100% (502/502), 108,03 КиБ | 0 ајтова/с, готово.
Укупно 502 (делта 303), поново коришћено 146 (делта 61)
даљинско: Компресовање изворних датотека... завршено.
даљински: Извор зграде: даљински:
```

```
даљински: ----> Питхон апликација откривена
даљински: ----> Инсталирање питхон-3.6.2 даљински:
----> Инсталирање пип даљинског управљача: ---->
Инсталирање захтева са пип-ом
...
даљински: ----> Удаљено откривање типова процеса:
Профил профила декларише типове -> ве
даљински:
даљински: ----> Компресија... даљински:
Урађено: 49,4М
даљински: ----> Покретање...
даљински: О јављен профилија 8
даљински: <аппнаме>.херокуапп.цом/ постављена на Хероку
даљински:
даљински: Верификација постављања... завршено.
На https://git.herokuapp.com/<аппнаме>.git * [нова
грана] мастер -> мастер
```

Апликација је сада распоређена и ради, али неће радити исправно јер команда деплои која иницијализује та еле азе података још није извршена. Хероку клијент може да покрене ову команду на следећи начин:

```
$ хероку рун флакс деплои  
Руннинг флакс деплои на <аппнаме>... уп, рун.3771 ( есплатно)  
ИНФО [алем иц.рунтиме.мигратион] Контекст импл ПостгресклиИmpl.  
ИНФО [алем иц.рунтиме.мигратион] Претпоставља се трансакцијски ДДЛ.  
...
```

Након што се креирају и конфигуришу та еле азе података, апликација се може поново покренути тако да почне чисто са ажурираном азом података:

```
$ хероку рестарт  
Поново покретање динос-а на <аппнаме>... готово
```

Апликација и сада тре ало да уде у потпуности распоређена и онлајн на [херокуапп.ком](https://<аппнаме>.herokuapp.com).

#### Прегледање дневника

апликације Хероку хвата излаз евидентије коју генерише апликација. Да исте видели садржај дневника, користите команду логс :

```
$ хероку дневники
```

Током тестирања, такође може ити згодно да се „прегледа“ датотека евидентије, што се може урадити на следећи начин:

```
$ хероку дневники -т
```

#### Примена надоградње Када

Хероку апликацију тре а надоградити, исти процес тре а поновити. Након што су све промене унете у Гит спремиште, следеће команде врше надоградњу:

```
$ хероку одржавање:он $ гит  
пусх хероку мастер $ хероку  
рун флакс распоређивање $  
хероку рестарт $ хероку  
одржавање: искључено
```

Опција одржавања доступна на Хероку ЦЛИ-у ће искључити апликацију током надоградње и приказаће статичну страницу која о авештава кориснике да ће се сајт ускоро вратити. Ово спречава кориснике да приступе апликацији док пролази кроз процес надоградње.

## Доцкер контејнери

Сада сте упознати са Херокуом, који је опција за постављање на прилично високом нивоу. У овом одељку ћете научити како да радите са контејнерима, а посебно са Доцкер платформом, која није толико аутоматизована као ПaaS, али пружа већу флексибилност и није везана за одређеног дољавчика о лака.

Контејнери су посебна врста виртуелних машина које раде на врху језгра оперативног система домаћина, за разлику од стандардних виртуелних машина које имају сопствено виртуелизовано језгро и хардвер. Пошто се виртуелизација зауставља на језгру, контејнери су много лакши и ефикаснији од виртуелних машина, али им је потреба на наменска подршка уградњена у оперативни систем. Линук кернел има пуну подршку за контејнере.

### Инсталирање Доцкер-

а Најпопуларнија платформа контејнера је [Доцкер](#), која има бесплатно Цомуунити Едитион (познато као Доцкер ЦЕ) и Ентерприсе Едитион засновано на претплати (Доцкер ЕЕ).

Доцкер се може инсталирати на три главна десктоп оперативна система, као и на сервере у објекту. Најлакши начин за развој и тестирање „контејнеризоване“ апликације је да инсталирате Доцкер ЦЕ на ваш развојни систем. За мацОС и Мицрософт Виндовс доступни су програми за инсталацију једним кликом из [Доцкер Сторе-а](#). Ова страница такође укључује упутства за инсталацију за ЦентОС, Федора, Дебијан и Убунту Линук дистрибуције.

Након што завршите инсталацију Доцкер ЦЕ на вашем систему, треће алоције и да удете у могућности да приступите доцкер команди са вашег терминала:

```
$ доцкер верзија
Клијент:
Верзија: 17.06.0-це
АПИ верзија: 1.30 Го
верзија: го1.8.3 Гит урезивање:
02ц1д87 Направљено: ОС/
Арџ: Пет Јун 23 21:31:53 2017
дарвин/амд64
```

```
Сервер:
Верзија: 17.06.0-це
АПИ верзија: 1.30 (минимална верзија 1.12)
Иди верзија: го1.8.3 Гит
урезивање: 02ц1д87
Направљено: пет, 23 Јул 2017 ОС/Арџ: линук/
амд64 Експериментално: истина
```



Доцкер за Виндовс захтева да Мицрософт Хипер-В функција уде омогућена. Инсталатор ће га оично омогућити уместо вас, али ако се чини да Доцкер не ради исправно након инсталације, прво трећа проверити стање хипервизора Хипер-В. Треће ало и да имате на уму да ће омогућавање Хипер-В на вашој Виндовс машини спречити рад других хипервизора (као што је Орацле ВиртуалБок).

Ако ваш систем не подржава Хипер-В виртуелизацију или вам је потреба но Доцкер решење које не чини неупотребљивим друге технологије виртуелизације, можда ћете желети да инсталирате [Доцкер Тоол ок](#), застарели Доцкер производ за Виндовс који је заснован на ВиртуалБок-у.

#### Прављење слике контејнера Први

задатак када радите са контејнерима је да направите слику контејнера за апликацију. Слика је снимак система датотека контејнера, који се користи као шаблон при покретању нових контејнера. Доцкер очекује да упутства за креирање слике будуће еђена у датотеки под називом Доцкер ле. [Пример 17-9](#) приказује Доцкер датотеку која гради апликацију представљену у овој књизи.

#### Пример 17-9. Доцкер ле: скрипта за прављење слике контејнера

[ИЗ питхон:3.6-алпине](#)

```
ЕНВ ФЛАСК_АПП фласки.py
ЕНВ ФЛАСК_ЦОНФИГ доцкер
```

```
РУН аддусер -Д фласки
УСЕР фласки
```

[ВОРКДИР /хоме/фласки](#)

Захтеви за ЦОПИ РУН питхон -м венв  
венв РУН венв/ ин/пип инсталл -р  
рекуирментс/доцкер.ткт

ЦОПИ апп апп
ЦОПИ миграције миграције ЦОПИ
фласки.py цонфиг.py оот.сх ./

```
# конфигурација времена извршавања
ЕКСПОСЕ 5000
ЕНТРИПОИНТ ["./ оот.сх"]
```

Команде изградње које се могу укључити у Доцкер датотеку су детаљно документоване у референца [Доцкер ле](#). У суштини, ово су команде за примену које инсталирају и конфигуришу апликацију у систему датотека контејнера, који је изолован од вашег система.

Команда ФРОМ је потре на у свим Доцкер датотекама да и се одредила слика основног контејнера од које тре а почети. У већини случајева, ово ће ити слика која је јавно доступна у Доцкер Ху -у, Доцкер-ово складиште слика контејнера. Репозиторијум садржи званичне слике за неколико верзија Питхон интерпретера. Ово су слике које имају основни оперативни систем са инсталацијама Питхоном. Слике су наведене са именом и ознаком. Име званичне Доцкер Ху Питхон слике је једноставно питхон. Различите ознаке које су доступне могу се видети на Доцкер Ху страници за слику. За Питхон слику, ознаке се користе за одређивање жељене верзије тумача и платформе.

За ову апликацију, 3.6 интерпретер изграђен на врху [Алпине Линук](#)-а користи се дистри уција. Алпине Линук је платформа која се очично користи у сликама контејнера з ог своје мале величине величина.



Верзије Доцкер-а за маџОС и Виндовс могу да покрећу контејнере засноване на Линук-у.

ЕНВ команда дефинише променљиве окружења за време извршавања . Ова команда узима два аргумента: име променљиве и њену вредност. Све променљиве окружења дефинисане овом командом иће доступне када се изврши контејнер заснован на овој слици. Овде је дефинисана променљива ФЛАСК\_АПП потре на за команду флакс , као и ФЛАСК\_ЦОНФИГ, што је име класе конфигурације коју апликација користи да се конфигурише када се покрене. Доцкер имплементација ће користити нову конфигурацију звану доцкер, имплементирану у класу ДоцкерЦонфиг као што је приказано у [Примеру 17-10](#). Ова нова конфигурациона класа наслеђује ПродуцтионЦонфиг и само конфигурише евидентирање да уде усмерено на стдерр, који Доцкер аутоматски хвата и излаже преко команде доцкер логс .

Пример 17-10. цон г.пи: Доцкер конфигурација

```
цласс ДоцкерЦонфиг(ПродуцтионЦонфиг):
    @цлассметод деф иниит_апп(цлс, апп):
        ПродуцтионЦонфиг.инит_апп(апп)
```

```
# логовање у
стдерр увоз
евиденције из евиденције импорт
СтреамХандлер файле_хандлер =
СтреамХандлер()
файлे_хандлер.сетЛевел(логгинг.ИНФО) апп.логгер.аддХандлер(файлे_хандлер)
```

```
конфигурација =
{ # ...
```

```
'доцкер': ДоцкерЦонфиг, #
...
}
```

Команда РУН извршава команду у контексту слике контејнера. У првом појављивању РУН-а, у контејнеру се креира фласки корисник. Команда аддусер је део Алпине Линук-а и доступна је у основној слици иза раној командом ФРОМ . -Д аргумент за аддусер потискује интерактивни упит за лозинку корисника.

Команда УСЕР ира корисника под којим ће се контејнер покренути, као и корисника за преостале команде у Доцкер фајлу. Доцкер подразумевано користи роот корисника, али се сматра да је редом праксом преласка на овог корисника када роот приступ није потребан.

Команда ВОРКДИР дефинише директоријум највишег нивоа у који ће апликација бити инсталirана. За ову апликацију се користи почетни директоријум за новокреiranог корисника фласки. Преостале команде у Доцкер фајлу ће се извршити са овим директоријумом као тренутним директоријумом.

Команда ЦОПИ копира датотеке из локалног система датотека у систем датотека контејнера. Директоријуми са захтевима, апликацијама и миграцијама се копирају у целини, а затим се копирају и фласки.пи, џон г.пи и нове датотеке оот.сx (о којима ће се ускоро расправљати).

Две додатне РУН команде креирају виртуелно окружење и инсталирају захтеве у њему. Наменски фајл са захтевима је креiran за Доцкер као захтеви/доцкер.ткт. Ова датотека увози све зависности из рекуирментс/џоммон.ткт и додаје Гунициорн, који ће се користити као веб сервер као у Хероку имплементацији.

Команда ЕКСПОСЕ дефинише порт на који ће апликација која ради унутар контејнера инсталирати свој сервер. Када се контејнер покрене, Доцкер ће мапирати овај порт у прави порт на главној машини, тако да контејнер може да прима захтеве из спољашњег света.

Последња команда је ЕНТРИПОИНТ. Ова команда одређује како да се изврши апликација када се контејнер покрене. Нова датотека оот.сx, копирана у горњи контејнер, користи се као скрипта за покретање. [Пример 17-11](#) приказује садржај ове датотеке.

Пример 17-11. оот.сx: скрипта за покретање контејнера

```
#!/bin/csh
соурсе венв/ин/активате
флак деплои екец гунициорн
- 0.0.0.0:5000 --аццес-логфиле - --еррор-логфиле - флакски:апп
```

Скрипта почиње активацијом венв виртуелног окружења које је креирано као део изградње. Затим покреће наред у деплои апликације , изграђену раније у овом поглављу и која се такође користи за примену Херокуа. Ово ће креирати нову азу података, надоградити је на најновију верзију и уметнути подразумеване улоге. Пошто ДАТАБАСЕ\_УРЛ променљива окружења није подешена, аза података ће користити СКЛите машину. Затим се покреће Гуницорн сервер који слуша на порту 5000. Доцкер хвата сав излаз из апликације и представља га као евиденције, тако да је Гуницорн конфигурисан да упише и своје приступне датотеке и датотеке евиденције грешака у стандардни излаз. Покретање Гуницорн-а са екец-ом чини да Гуницорн процес преузме процес који покреће датотеку оот.сх. Ово је учињено зато што Доцкер посе ну пажњу посвећује процесу који покреће контејнер, и очекује да ће он или главни процес током његовог животног века. Када се овај процес заврши, завршава се и контејнер.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 17д да исте проверили ову верзију апликације.

Слика контејнера за Фласки сада може да се направи на следећи начин:

```
$ доцкер уилд -т фласки: најновије .
Слање контекста изградње Доцкер демону 51.08МБ Корак
1/14: ИЗ питхон:3.6-алгине
--> ab ea 4fa70
...
Успешно изграђен 930e17a89 42
Успешно означен фласки:латест
```

Аргумент -т за доцкер уилд даје име и ознаку слици контејнера, одвојене двотачком. Најновије име ознаке се оично користи за најновију верзију слике контејнера . Тачка на крају команде изградње поставља тренутни директоријум као директоријум највишег нивоа током изградње. Доцкер ће тражити Доцкерле у овом директоријуму, а такође ће учинити датотеке у овом директоријуму и свим поддиректоријумима доступним за додавање у слику контејнера.

Као резултат успешне команде за прављење доцкер -а, изграђена слика контејнера се чува у локалном спремишту слика. Команда доцкер имагес приказује садржај спремишта слика на вашем систему:

\$ доцкер имагес			
ОЗНАКА	ИД СЛИКЕ НАПРАВЉЕН	СИЗЕ	
РЕПОЗИТОРИЈА фласки	930e17a89 42 пре 5 минута 127МБ 88.7МБ		
најновији питхон:3.6-алгине	аб ea 4fa70	пре 3 недеље	

Овај списак укључује управо направљену фласки:најновију слику, као и основну слику тумача за Питхон 3.6 наведену у ФРОМ Доцкер фајлу, коју Доцкер преузима и инсталира као део израде.

#### Покретање контејнера Када

се направи слика контејнера за апликацију, остаје само да се покрене. Доцкер рун команда чини ово веома једноставним задатком:

```
$ доцкер рун --наме фласки -д -п 8000:5000 \
-e СЕЦРЕТ_КЕИ=57д40ф677афф4д8д96дф97223ц74д217 \ -e
МАИЛ_УСЕРНАМЕ=<ваше-гмаил-корисничко име> \ -e
МАИЛ_ПАССВОРД=<ваша-гмаил-лозинка> фласки:најновија
```

Опција --наме даје назив контејнеру. Именовање контејнера је опционо; ако име није дато, Доцкер га генерише користећи насумично ода ране речи.

Опција -д покреће контејнер у одвојеном режиму, што значи да ће контејнер радити у позадини на вашем систему. Контејнер који није одвојен ради као задатак у првом плану прикачен сесији конзоле.

Опција -п мапира порт 8000 у систему домаћина у порт 5000 унутар контејнера.

Доцкер о ез еђује флекси илност мапирања портова контејнера на ило који порт у систему домаћина. Ово мапирање омогућава да две или више инстанци исте слике контејнера раде на различитим портовима домаћина, док свака инстанца користи сопствени виртуелизовани порт 5000.

Опција -е дефинише променљиве окружења које ће постојати у контексту контејнера, поред свих променљивих дефинисаних у време изградње помоћу ЕНВ команде у Доцкер фајлу. Вредност додељена променљивој СЕЦРЕТ\_КЕИ о ез еђује да су корисничке сесије и токени потписани јединственим кључем који је веома тешко погодити. Тре ало и да генеришете сопствени јединствени кључ за ову променљиву. Вредности за променљиве МАИЛ\_УСЕРНАМЕ и МАИЛ\_ПАССВОРД конфигуришу слање е-поште преко Гмаил услуге. За примену у производњи која користи другог до ављача услуга е-поште, променљиве МАИЛ\_СЕРВЕР, МАИЛ\_ПОРТ и МАИЛ\_УСЕ\_ТЛС такође тре а да уду дефинисане.

Последњи аргумент у доцкер рун команди је слика контејнера и ознака за извршење. Ово и тре ало да се подудара са именом и ознаком датим као опција -т команди за изградњу доцкер -а.

Када се контејнер покрене у позадини, доцкер рун команда штампа Ид контејнера на конзоли. Ово је 256- итни јединствени идентификатор штампан у хексадецималној нотацији. Овај Ид се може користити у свим командама које захтевају референцу на контејнер (у пракси, само првих неколико карактера Ид-а тре а да се наведе, тако да се контејнер може јединствено идентификовати).

Да исте потврдили да је контејнер покренут, може се користити доцкер ps команда:

\$ доцкер ps	СЛИКА ИД КОНТЕЈНЕРА	ЦРЕАТЕД	СТАТУС	ЛУКЕ	НАМЕС
	71357ee776ae	фласки:латест 4 сеџ аго Уп 8 сеџ 0.0.0:8000->5000/тцп	фласки		

Пошто је контејнер сада покренут и покренут, можете приступити контејнеризованој апликацији на порту 8000 вашег система, ило локално као `хттп://лоцалхост:8000` или са ило ког другог рачунара у мрежи као `хттп://<ип- адреса>: 8000.`

Да исте зауставили овај контејнер, користите доцкер стоп команду:

```
$ доцкер стоп 71357ee776ae  
71357ee776ae
```

Команда стоп зауставља контејнер, али га не уклања из система. Да исте га уклонили, користите команду доцкер rm :

```
$ доцкер rm 71357ee776ae  
71357ee776ae
```

Ове две операције се могу коминовати у једну помоћу доцкер rm -f:

```
$ доцкер rm -f 71357ee776ae  
71357ee776ae
```

Провера покренутог контејнера Када изгледа да

се контејнер не понаша, можда ће ити потре но да га отклоните. Најочигледнији механизам за отклањање грешака је додавање изјава за евидентирање у апликацију, а затим праћење покренутог контејнера помоћу команде доцкер логс .

У неким ситуацијама, међутим, можда и ило згодније отворити сесију љуске на покренутом контејнеру како и се могла пажљивије прегледати. Команда доцкер екец ово омогућава:

```
$ доцкер екец -ит 71357ee776ae сх
```

У овом примеру, Доцкер ће отворити сесију љуске са сх (уникс школјком) ез прекидања контејнера. Опције -ит повезују терминалску сесију из које се издаје команда са новим процесом, тако да љуском може да се ради интерактивно. Ако контејнер укључује друге, напредније школке као што су асх или чак Питхон интерпретер, они се такође могу користити.

Уо ичајена стратегија при решавању пролема са контејнерима је креирање посечне слике учитане додатним алатима као што је програм за отклањање грешака који се касније може позвати из љуске седница.

Пре ацивање слике вашег контејнера у спољни регистар Имати слику контејнера локално је згодно када се развија и тестира апликација, али када сте спремни да поделите слику са другима, морате да је гурнете на спољни сервер регистра.

Доцкер Ху регистар је Доцкер-ово спремиште слика, погодна услуга на којој можете да хостујете своје слике. Бесплатни Доцкер Ху налог вам омогућава да чувате неограничен број слика јавних контејнера, али само једну приватну слику. Плаћени планови повећавају број приватних слика које можете да хостујете. Да исте креирали свој Доцкер Ху налог, идите на <https://xy.dockcer.com>.

Када имате Доцкер Ху налог, можете се пријавити на њега из командне линије помоћу доцкер логин команде:

```
$ доцкер логин
```

Пријавите се са својим Доцкер ИД-ом да исте гурнули и повукли слике из Доцкер Ху -а.

Корисничко име: <иоур-доцкерху -усернаме>

Лозинка: <иоур-доцкерху -пассворд> Пријава

је успела



Да исте се пријавили у спремиште слика контејнера које није Доцкер Ху, проследите адресу свог спремишта као аргумент за пријаву на Доцкер.

Слике локалних контејнера до ијају једноставно име. Да исте се припремили за прослеђивање слике у Доцкер Ху, име слике мора имати префикс имена Доцкер Ху налога и косу црту као сепаратор. Фласки:најновијој слици која је раније направљена може се дати секундарно име правилно форматирано за прослеђивање у Доцкер Ху помоћу команде доцкер таг -а:

```
$ доцкер таг фласки:латест <иоур-доцкерху -усернаме>/фласки:латест
```

Да исте отпремили слику у Доцкер Ху, користите доцкер пусх команду:

```
$ доцкер пусх <иоур-доцкерху -усернаме>/фласки:латест
```

Слика контејнера је сада јавно доступна и свако може да покрене контејнер на основу ње помоћу доцкер рун команде:

```
$ доцкер рун --наме фласки -д -п 8000:5000 \ <иоур-доцкерху -усернаме>/фласки:латест
```

Коришћење екстерне азе

података Један недостатак начина на који је Фласки ио примењен као Доцкер контејнер је то што подразумевана СКЛите аза података живи у истом контејнеру као и апликација. Ово отежава извођење надоградње, јер када се покренути контејнер заустави, аза података нестаје са њим.

Болији приступ је да се сервер азе података хостује одвојено од контејнера апликације. То чини надоградњу апликације уз очување азе података лаким задатком, јер је све што је потребе но је да се контејнер апликације замени новим.

Доцкер промовише модуларни приступ изградњи апликације, у којој се сваки сервис налази у сопственом контејнеру. Доступне су слике јавних контејнера за МиСКЛ, Постгрес и многе друге сервере аза података. Доцкер рун команда се може користити за примену или које од њих директно на ваш систем. Следећа команда поставља МиСКЛ 5.7 сервер азе података на ваш систем:

```
$ доцкер рун --наме мискл -д -е МИСКЛ_РАНДОМ_РООТ_ПАССВОРД=да \ -е  
МИСКЛ_ДАТАБАСЕ=фласки -е МИСКЛ_УСЕР=фласки \ -е  
МИСКЛ_ПАССВОРД=<лозинка> азе података \ мискл/мискл7.
```

Ова команда креира контејнер под именом мискл који ради у позадини. Опција -е додељује неколико променљивих окружења које овај контејнер узима као конфигурацију.

Ове и многе друге варијаље су документоване на страници Доцкер Ху за МиСКЛ слику. Претходна команда конфигурише азу података са насумично генерираном роот лозинком (користите доцкер евидентију мискл одмах након покретања контејнера да исте видели додељену лозинку у евидентији), и са потпуно новом азом података која се зове фласки која је конфигурисана да јој приступи корисник назван и фласки. Морате да о је едите је једну лозинку за овог корисника као вредност за променљиву окружења МИСКЛ\_ПАССВОРД.

Да и се могао повезати са МиСКЛ азом података, СКЛАлцхеми захтева да се инсталира подржани МиСКЛ клијентски пакет као што је пимискл. Овај пакет се може додати у датотеку са захтевима доцкер.ткт.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхецкоут 17е да исте проверили ову верзију апликације.

Промена направљена у датотеци Захтеви/доцкер.ткт захтева да се слика контејнера поново направи:

```
$ доцкер уилд -т фласки: најновије .
```

Ако још увек користите претходни контејнер апликације, зауставите га и уклоните помоћу доцкер рм -ф.  
Затим покрените нови контејнер са ажурираном апликацијом:

```
$ доцкер рун -д -п 8000:5000 --линк мискл:д сервер \
-е ДАТАБАСЕ_УРЛ=мискл+пимискл://фласки:<лозинка- азе података>@д сервер/фласки \ -е
МАИЛ_УСЕРНАМЕ=<ваше-гмаил-корисничко име> -е МАИЛ_ПАССВОРД=<ваша-гмаил-лозинка>
\ фласки:најновија
```

Овде су приказана два додатка доцкер рун команда. Опција --линк конфигурише везу између новог контејнера и другог постојећег. Аргумент за --линк састоји се од два имена одвојена двотачком: назив изворног контејнера или ИД и псевдоним за тај контејнер у контејнеру који се креира. У овом примеру изворни контејнер је мискл, контејнер азе података је покренут раније. Овај контејнер ће ити доступан у новом Фласки контејнеру са д сервер именом хоста.

Да и се довршила конфигурација, додаје се променљива окружења ДАТАБАСЕ\_УРЛ , са УРЛ-ом везе која указује на фласки азу података у мискл контејнеру. Псевдоним д сервер се користи као хост азе података, јер Доцкер осигурува да се ово име разреши у ИП адресу повезаног контејнера. Вредност променљиве окружења МИСКЛ\_ПАССВОРД постављена у мискл контејнеру такође мора ити укључена у УРЛ везе за овај контејнер. Вредност ДАТАБАСЕ\_УРЛ замењује подразумевану СКЛите азу података, тако да ће овом једноставном променом контејнер ити конфигурисан да се повеже са МиСКЛ азом података.



Доцкер Ху спремиште је златни рудник веома корисних апликација и услуга које су упаковане и спремне за употребу у Доцкер окружењу, ило самостално или као основне слике за ваше сопствене контејнере. Видећете да све врсте пројектата (укључујући азе података, ве сервере, алансере оптерећења, програмске језике, оперативне системе и још много тога) нуде званичне слике.

#### Оркестрација контејнера са Доцкер Цомпосе Контејнерске апликације

се о ично састоје од неколико покренутих контејнера. У претходном одељку сте видели да главна апликација и сервер азе података раде у независним контејнерима. Како апликација постаје све сложенија, увек ће јој тре ати више контејнера. Неке апликације ће захтевати додатне услуге, као што су редови порука или кеш меморије. Друге апликације могу искористити предности архитектуре микросрвиса и имати дистри уирану структуру са неколико мањих су

апликације, од којих свака ради у свом контејнеру. Апликације које морају да поднесу велика оптерећења или морају да уду толерантне на грешке ће желети да се смање тако што ће покренути неколико инстанци иза алансера оптерећења.

Како се рој контејнера који су део апликације повећава, задатак управљања и координације свих ових контејнера ће постати много тежи ако се користи само Доцкер. Оквири оркестрације контејнера изграђени на врху Доцкер-а помажу у овом задатку.

Скуп алата Цомпосе пружа основну функционалност оркестрације, укључену у инсталацију Доцкер-а. Са Цомпосе, контејнери који су део апликације су описаны у конфигурационој датотеци, која се оично зове доцкер-цомпосе.имл. Команда доцкер цомпосе тада може покренути све контејнере повезане са апликацијом помоћу једне команде.

**Пример 17-12** показује датотеку доцкер-цомпосе.имл која представља контејнерски Фласки заједно са његовим МиСКЛ сервисом.

Пример 17-12. доцкер-цомпосе.имл: конфигурација компоновања

верзија: '3' услуге:

фласки:  
уилд: .

портови: - "8000:5000"

енв\_филе: .енв

линкови:

- мискл:д сервер

рестарт: увек

мискл: имаге: "мискл/

мискл **-сервер:5.7** енв\_филе: .енв-

мискл рестарт: увек

Ова датотека је написана у ИАМЛ-у, који је чист и једноставан формат који може представљати хијерархијске структуре које се састоје од мапа кључ-вредност и листа. Кључ верзије одређује која се верзија Цомпосе користи, а кључ услуга дефинише контејнере апликације као њене потомке. У случају Фласкија, ово су две услуге под називом фласки и мискл.

За услуге као што је фласки, које су изграђене као део апликације, поткључеви специфицирају аргументе који се дају наредама за изградњу и покретање доцкер-а. Кључ за изградњу специфицира директоријум за прављење, где се налази Доцкер датотека.

Кључ портова одређује мапирања мрежних портова. Енв\_филе кључ је згодан начин да се дефинише неколико варија ли окружења које су потребне контејнеру. Кључ линкс успоставља везу са МиСКЛ контејнером тако што га излаже са именом хоста д сервер. Кључ за поновно покретање постављен на увек пружа једноставан начин за Доцкер да аутоматски поново покрене контејнер ако неочекивано изађе. Датотека .енв за ову примену треба да садржи следеће променљиве:

```

Фласк_апп = Фласки.пи
Фласк_конфиг = Доцкер
Сеџрет_Кеи = 3128Б4588Е7Ф4305Б5501025Ц13ЦЕЦА5
маил_усернаме = <Иоур-Гмаил-корисничко име>
маил_палл-лозинка = <иоус-гмаил-лозинка> Дата ace_url
= МиСКЛ + пимискл: // Фласки: < аза података-лозинка> @ д сер / фласки

```

Мискл услуга има једноставнију структуру, јер је ово услуга која се покреће са слике залиха која не захтева корак изградње. Кључ слике одређује име и ознаку слике контејнера који ће се користити за ову услугу. Као и са командом за покретање доцкер-а , Доцкер ће преузети ову слику из регистра слика контејнера. Енв\_филе и кључеви за рестарт су слични онима који се користе у фласки контејнеру. О ратите пажњу на то како се променљиве окружења за МиСКЛ контејнер чувају у засе ној датотеки под називом .енв-мискл. Иако и ило лакше додати променљиве окружења које су потре не свим контејнерима у .енв датотеку, до ра је пракса спречити да један контејнер има приступ тајнама другог. Датотеки .енв-мискл тре а дефинисати следеће променљиве окружења:

```

МИСКЛ_РАНДОМ_РООТ_ПАССВОРД=да
МИСКЛ_ДАТАБАСЕ=фласки
МИСКЛ_УСЕР=фласки
МИСКЛ_ПАССВОРД=<лозинка- азе података>

```



Датотеке .енв и .енв-мискл садрже лозинке и друге осетљиве информације, тако да их никада не тре а додавати у контролу извора.



Комплетна референца за датотеку доцкер-цомпосе.имл налази се на „[Доцкер ве локацији](#)“.

Типичан про лем са оркестирираним системима је тај што се контејнери покрећу погрешним редоследом—или исправним редоследом, али ез давањаово времена контејнерима за основне услуге да се покрену и иницијализују пре покретања контејнера вишег нивоа који зависе од њих. У случају Фласкија, мискл контејнер мора да се покрене први, тако да аза података уде покренута када се фласки контејнер покрене. Затим се може повезати са азом података, применити миграције азе података и коначно покренути ве сервер.

Цомпосе ће покренути мискл и фласки контејнере у правом редоследу, јер ће открити зависност између њих из кључа линкова у фласки контејнеру. Али Цомпосе неће чекати да се МиСКЛ покрене, што може потрајати неколико секунди.

Приликом пројектовања дистри уираних система, до ра је пракса имплементирати поновне покушаје у свим конекцијама на спољне услуге. [Пример 17-13](#) показује како оот.сх скрипта то

стартовања фласки контејнер се може учинити да уснијим поновним покушајем команде флак деплои , која понавља надоградњу азе података док не успе.

Пример 17-13. оот.сх: чека да се аза података покрене

```
#!/ ин/сх
извор венв/ ин/актив

док је истинито;
    применити флак
    ако [[ "$?" == "0" ]]; затим сломити

фи
ецх Команда распоређивања није успела, поновни покушај за 5 секунди...
спавање 5 завршено
```

екец гуницорн - :5000 --аццес-логфиле - --еррор-логфиле - фласки:апп

Покретањем флак деплои унутар петље за поновни покушај, контејнер ће моби да толерише грешке з ог тога што услуга азе података није одмах спремна да прихвати захтеве.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит цхецкоут 17ф да исте проверили ову верзију апликације. Такође се уверите да су датотеке окружења .енв и .енв-мискл креирани и попуњене вредностима тачним за ваше окружење.мент.

Сада када је конфигурација Цомпосе завршена, апликација се може покренути командом доцкер-цомпосе уп :

```
$ доцкер-цомпосе уп -д -- уилд
```

Опција -- уилд за доцкер-цомпосе уп означава да корак изградње тре да се покрене пре покретања апликације. Ово ће проузроковати да се направи слика фласки контејнера. Након што се слика креира, мискл и фласки контејнери ће или покренути тим редоследом. Опција -д покреће контејнере у одвојеном режиму, као и са појединачним контејнером. Након неколико секунди, апликација и тре ало да ради у позадини и тре ало и да удете у могућности да се повежете са њом на хттп://лоцалхост:8000.

Цомпосе консолидује евиденцију из свих контејнера у један ток, што можете видети помоћу команде доцкер-цомпосе логс :

```
$ доцкер-цомпосе евиденције
```

Или, ако желите да стално пратите ток евиденције:

```
$ доцкер-цомпосе логс -ф
```

Команда доцкер-цомпосе пс приказује резиме свих контејнера апликација који су покренути и њихово стање:

\$ доцкер-цомпосе пс	Име	Цоманд	Стање	Портс
фласки_фласки_1 ./ оот.сх фласки_мискл_1	Горе /ентрипоинт.сх мисклд Горе		0.0.0:8000->5000/тцп 3306/тцп, 33060/тцп	

Да исте надоградили апликацију на нову верзију, једноставно унесите потре не измене у њу и поновите наред у доцкер-цомпосе уп која је претходно коришћена за њено покретање. Цомпосе ће поново изградити контејнер апликације ако се нешто промени, а затим ће заменити старији контејнер новим.

Да исте зауставили апликацију, користите команду доцкер-цомпосе довн или доцкер-цомпосе рм --стоп --форце ако желите да уклоните и заустављене контејнере.

Чишћење старих контејнера и слика Док радите са

контејнерима, ваш систем ће увек акумулирати старе контејнере или слике које више нису потре не. До ра је идеја да их рутински прегледате и очистите како не и заузимали простор на систему.

Да исте видели листу контејнера у систему, користите следећу команду:

```
$ доцкер пс -а
```

Ово ће приказати контејнере који су покренути и контејнере који су заустављени, али су још увек у систему. Да исте из рисали контејнере са ове листе, користите команду доцкер рм -ф и наведите имена или ИД-ове за уклањање:

```
$ доцкер рм -ф <име-или-ид> <име-или-ид> ...
```

Да исте видели листу слика контејнера усклађиштених у вашем систему, користите команду доцкер имагес . Ако постоје слике које желите да уклоните, то можете учинити помоћу команде доцкер рми .

Неки контејнери креирају виртуелне волумене на главном рачунару који се користе за складиштење ван система датотека контејнера. Слика МиСКЛ контејнера, на пример, ставља све датотеке азе података у волумен. Можете да видите листу свих додељених волумена у вашем систему помоћу доцкер волумен лс. Да исте уклонили волумен који се не користи, користите доцкер волумен рм.

Ако више волите аутоматско чишћење, команда доцкер система пруне --волумес ће уклонити све некоришћене слике или волумене, као и све заустављене контејнере који су још увек у систему.

## Коришћење Доцкер-а у производњи

Многи људи сматрају Доцкер само платформом за развој и тестирање. Иако се технике представљене у претходним одељцима могу користити за примену апликација на производним серверима који користе Доцкер, постоје нека ограничења и ез едносни про леми које тре а узети у о зир:

### Надгледање и упозорење

Шта се дешава ако се апликација у контејнерима сруши? Доцкер може поново покренути контејнер који неочекивано излази, али неће надгледати ваше контејнере, нити ће слати упозорења када се понашају неуредно.

### Доцкер

за евидентирање одржава поседан ток евиденције за сваки контејнер. Цомпосе ово по ољшава нудећи консолидовани ток, али ез могућности дуготрајног складиштења или претраживања и филтрирања.

### Управљање тајнама

Конфигурисање лозинки и других акредитива преко променљивих окружења није ез едно, пошто Доцкер излаже унапред конфигурисане променљиве окружења преко команде доцкер инспект или преко свог АПИ-ја.

### Поузданост и скалирање

Да и се помогло у толеранцији грешака, или да и се прилагодили растућим захтевима оптерећења, неопходно је покренути неколико инстанци апликације на неколико хостова и иза једног или више алансера оптерећења.

Ова ограничења се генерално решавају разрађенијим оквирима оркестрације изграђеним на врху Доцкер-а или других времена извршавања контејнера. Оквири као што су Доцкер Сварм (сада утврђен у Доцкер), Апаџе Месос и Куернетес су дошао из ар за изградњу ресурсног постављања контејнера.

## Традиционалне примене

До сада сте видели како Хероку и Доцкер управљају применама. Да и смислили овај преглед стратегија примене, овај одељак ће описати традиционалну опцију хостовања, која укључује куповину или изнајмљивање сервера, и ли физичког или виртуелног, и ручно подешавање свих потребних компоненти на њему. Ово је очигледно најзахтевнија опција од свих, али може ити згодна опција када имате терминалски приступ хардверу производног сервера. Следећи одељци ће вам дати идеју о послу који је укључен.

## Подешавање сервера

Постоји неколико административних задатака који се морају остварити на серверу да и могао да хостује апликације:

- Инсталирајте сервер базе података као што је МиСКЛ или Постгрес. Коришћење СКЛите базе података је такође могуће, али се не препоручује за производни сервер због ројних ограничења у погледу модификације постојећих шема базе података.
- Инсталирајте Mail Транспорт Агент (MTA) као што је Сендмаил или Пост када исте слали е-пошту корисницима. Коришћење Гмаил-а у производној апликацији није могуће, јер ова услуга има веома рестриктивне квоте и посебно за праћење комерцијалну употребу у условима коришћења услуге.
- Инсталирајте веб сервер спреман за производњу као што је Гуницорн или уВСГИ. • Инсталирајте услужни програм за праћење процеса, као што је Супервизор, који одмах поново покреће веб сервер ако се сруши или након што се хост укључи. • Инсталирајте и конфигуришите SSL сертификат да исте омогућили везу едан ХТТП. • (Опционално, али веома препоручљиво) Инсталирајте фронт-енд реверсе прокси веб сервер као што је Нгинкс или Апаџе. Овај сервер је конфигурисан да директно служи статичке датотеке и прослеђује захтеве апликације на веб сервер апликације, који слуша на приватном порту на локалном хосту.
- Овде ћете видети како укључује неколико задатака који имају за циљ смањење рањивости на серверу, као што су инсталација заштитних зидова, уклањање некоришћеног софтвера и услуга итд.



Уместо да ручно обављате ове задатке, креирајте скриптовану примену користећи оквир за аутоматизацију као што јеAnsible, Цхеф или Пуплет.

Увоз варијаја или окружења Слично као код Хероку-а и

Доцкер-а, апликација која ради на самосталном серверу се ослања на одређена подешавања као што су УРЛ везе са азом података, акредитиви сервера е-поште, итд. која се налазе у варијајама окружења.

Пошто не постоји Хероку или Доцкер за конфигурисање ових променљивих пре него што апликација почне, процедура за постављање променљивих зависи од платформе и коришћених алатака. Да и конфигурација променљивих окружења ила лакша и уједначена на платформама за примену, лок кратког кода у [примеру 17-14](#) увози у окружење .env датотеку сличну оној која се користи са хероку лоцал и доцкер-ом.

компонујте команде, користећи Питхон пакет који се зове питхон-дотенв који треба да се инсталира са pip-ом. Ово се ради у фласки.пи пре него што се инстанца апликације креира, тако да до тренутка увоза конфигурације ове променљиве буду доступне у окружењу.

Пример 17-14. фласки.пи: увоз окружења из .енв ле

```
импорт os
фром дотенв импорт лоад_дотенв

дотенв_патх = os.патх.јоин(os.патх.дирнаме(__file__), '.енв') ако
os.патх.екистс(дотенв_патх): лоад_дотенв(дотенв_патх)
```

Датотека .енв може да дефинише променљиву FLASK\_ЦОНФИГ којаира конфигурацију коју ће користити, везу ДАТАБАСЕ\_УРЛ, акредитиве сервера е-поште, итд. Као што је раније о јашњено, .енв фајл не треба да се додаје у контролу извора због осетљиве природе неких од ставке у њему.



Ако сте креирали .енв датотеку за коришћење са Хероку-ом или Доцкер-ом, прегледајте је и прилагодите је на одговарајући начин, јер ће са управо направљеним изменама апликација увести променљиве дефинисане у овој датотеци за све конфигурације.

#### Подешавање евидентирања

За сервере засноване на Уник-у, евидентирање се може послати демону сислог. Нова конфигурација поседује за Уник може бити креирана као подкласа ПродуџионЦонфиг, као што је приказано у Примеру 17-15.

Пример 17-15. цон.г.пи: Уник пример конфигурације

```
класа УникЦонфиг(ПродуџионЦонфиг):
    @классметод
    деф инит_апп(цлс, апп):
        ПродуџионЦонфиг.инит_апп(апп)

        # лог то сислог
        импорт логгинг
        фром логгинг.хандлерс импорт СисЛогХандлер
        сислог_хандлер = СисЛогХандлер()
        сислог_хандлер.сетЛевел(логгинг.ВАРНИНГ)
        апп.логгер.аддХандлер(сислог_хандлер)
```

Са овом конфигурацијом, евидентирање апликација ће бити уписане у конфигурисану датотеку сислог порука, о чијно /вар/лог/мессажес или /вар/лог/сислог у зависности од Линук дистри-

утион. Сервис сислог се може конфигурисати да пише засе ну датотеку дневника за евиденције апликације или да пошаље евиденције на другу машину ако се жели.



Ако сте клонирали Гит спремиште апликације на ГитХу -у, можете покренути гит џхеџкоут 17г да исте проверили ову верзију апликације.

## ГЛАВА 18

### Додатна средства

Готово сте завршили са овом књигом. Честитам! Надам се да су вам теме које сам покрио дале солидну основу да почнете да правите сопствене апликације са Фласком. Примери кода су отвореног кода и имају дозвољену лиценцу, тако да сте до рдошли да користите онолико мого кода колико желите да поставите своје пројекте, чак и ако су комерцијалне природе. У овом кратком завршном поглављу желим да вам дам листу додатних савета и ресурса који и могли ити корисни док наставите да радите са Фласком.

#### Коришћење интегрисаног развојног окружења (ИДЕ)

Развијање Фласк апликација у интегрисаном развојном окружењу (ИДЕ) може ити веома згодно, пошто функције као што су довршавање кода и интерактивни де агер могу значајно у рзати процес кодирања. Неки од ИДЕ-ова који до ро функционишу са Фласком наведени су овде:

##### ПиЦхарм

ИДЕ из ЈетБраинс-а са Цомунити (есплатним) и Профессионал (плаћеним) издањима, о а компати илна са Фласк апликацијама. Доступно на Линук-у, мацОС-у и Виндовс-у.

##### Висуал Студио Цоде

ИДЕ отвореног кода из Мицрософта. Мора ити инсталiran Питхон додатак треће стране да исте имали приступ функцијама довршавања кода и отклањања грешака са Фласк апликацијама. Доступно на Линук-у, мацОС-у и Виндовс-у.

##### ПиДев

ИДЕ отвореног кода заснованог на Еклипсе-у. Доступно на Линук-у, мацОС-у и Виндовс-у.

## Пronалажење флакс екстензија

Примери у овој књизи ослањају се на неколико проширења и пакета, али има много више оних који су такође корисни и о којима се није расправљало. Следи кратка листа неких додатних пакета које вреди истражити:

- **Флакс-Бајел:** Подршка за интернационализацију и локализацију •

**Марсхмаллов:** Серијализација и десеријализација Питхон објеката, корисни за представљање АПИ ресурса • **Целер:** Ред задатака за обраду позадинских послова • **Фрозен-Флакс:** Конверзија Флакс апликације у статичну веб локацију • **Флакс-Деј угТоол ар:** Алати за отклањање грешака у прегледачу • **Флакс-активе:** Спајање, минимизирање и компајлирање ЦСС и Јаваскрипт средстава • **Флакс-Сессион:** Алтернативна имплементација корисничких сесија које користе серверску страну

складиште

- **Флакс-СоцкетИО:** Имплементација Соцкет.ИО сервера са подршком за Веб Соцкет и дуготрајно испитивање

Ако функционалност која вам је потребна за ваш пројекат није покривена ниједним од проширења и пакета поменутих у овој књизи, онда и ваше прво одредиште за тражење додатних екстензија треба да буде званични [регистар проширења Флакс](#). Друга добра места за претрагу су [Питхон Пацкаче Индекс](#), [ГитХуб](#), и [Бит уцкет](#).

## Доносијање помоћи

Ако дођете до тачке у којој сте блокирани проблемом који не можете сами да решите, имајте на уму да постоји заједница Флакс програмера попут вас која ће вам радо помоћи.

Одлично место за постављање питања о Флаксу или ило којим сродним проширењима је [Стацк Оверфлов](#). Други програмери који виде ваше питање и знају како да одговоре објавиће своје одговоре, за које се гласало горе или против у складу са њиховим квалитетом. Ви, као власник питања, тада можете изабрати најбољи одговор. Сва питања и њихови одговори остају на сајту и појављују се у резултатима претраге. Дакле, постављањем питања на овој платформи помажете у повећању прикупљања информација о Флаксу.

Реддит такође има пријатељски [субреддит посвећен Флаксу](#) где можете постављати питања.

Конечно, ако користите ИРЦ, #поцоо канал на Фрееноде-у посећују Флакс програмери свих нивоа који вам могу помоћи један на један са вашим проблемом.

## Укључивање у Фласк

Флак не и ио тако сјајан као што јесте ез посла који је о авила његова заједница програмера. Пошто сада постаете део ове заједнице и имате користи од рада толиког роја волонтера, тре ало и да размислите о проналажењу начина да нешто вратите. Ево неколико идеја које ће вам помоћи да почнете:

- Прегледајте документацију за Флак или ваш омиљени сродни пројекат и пошаљите исправке или по ољшања.
- Преведите документацију на нови језик. • Одговарајте на питања на сајтовима за питања и одговоре као што је [Стацк Оверфлов](#). • Разговарајте о свом раду са својим вршњацима на састанцима групе корисника или конференцијама. • Допринесите исправке грешака или по ољшања пакета које користите. • Напишите нове екstenзије за Флак и пустите их као отворени код. • О јавите своје апликације као опен соурце.

Надам се да ћете се одлучити за волонтирање на један од ових начина, или ило који други који вам је од значаја. Ако јесте, хвала вам!

## Индекс

- Функција прекида, 22, 231  
 апсолутна УРЛ адреса (у везама),  
 37 потврда налога, 118-125 генерисање  
     токена за потврду са својим- опасним, 118 слање е-  
     порука са потврдом, 120-124 управљање  
     налогом, 125 додела улога корисницима, 135  
     активација, виртуелна окружења, 4 улоге  
         администратора, 127 додељивање, 131  
     администратор, уређивач корисничког профила за,  
     143-145 декоратор за админ\_рекуиред, 145 закачивање  
         афтер\_апп\_рекуест , 238 кука афтер\_рекуест, 20  
     инстанца апликације, 7  
     структуре апликације, основна, 7-23  
         опције командне линије, 15  
         комплетних примера апликације, 9  
         режим за отклањање грешака, 13  
         развојни сервер, 10 динамичких пута, 12  
     екстензије за фласк, 23  
     апликације за иницијализацију,  
     7 циклуса захтев-одговор, 17-22 руте  
     и функције прегледа, 8  
     апликације, велика, структура, 85-97 пакет  
     апликација, 88-92 имплементација  
         функционалности апликације у нацрту, 90-92  
         користећи фа рику апликације, 88-89 скрипта  
         апликације, 93 опције конфигурације, 86-88  
         подешавање аз података, 96 структура пројекта ,  
         85 датотека са захтевима, 93 покретање апликације,  
         97 јединичних тестова, 94 декоратор апп\_errorхандлер,  
         91, 205 асоцијацијских та ела ( аза података), 65,  
         172 та ела пратилаца као модел, 174  
         аутх.логин\_рекуиред декоратор, 208 потврда  
         аутентификације-1-15 налога , 118-125 управљање  
     налогом, 125 креирање нацрта за, 105 Фласк екстензије  
     за, 101 Фласк-Майл Гmail налог, 80 у АПИ нацрту, 204  
     ресурси, 200 ве  
     сервиса за верзије, 202 РЕСТфул  
     ве сервиса са Фласк, 203-218 апп.адд\_урл\_руле  
     метод, 8, 19 апп.чили.комманд декоратор, 96 апп.ционфиг  
     о јекат, 44 апп.рун метод, 11 апп.схелл\_онтекст\_процесор  
     децоратор 72 контекст апликације, 18 и е-пошта на  
     позадинској нити, 83 директоријум апликације,  
     креирање, 2

регистрација нових корисника, 115-118  
сигурност лозинком, 102-105 на ази  
токена, 208-210 аутентификација корисника  
са Флакс-Логин, 107-115

са Флакс-ХТТПАутх, 206 са Хероку  
налогом, 245 оквира за  
автоматизацију, 271 аватара, 146-149  
аватара на страници профиле, 147  
автора лог постова, 153

Аргументи стрингова упита Граватар, 146  
Граватар УРЛ генерирање, 146  
ABC ЕЦ2, 243

## Б

позадински послови, за слање е-поште, 84  
позадинска нит, е-пошта укључена, 83 асх, 2  
ефоре\_app\_рекуест декоратор, 122, 138  
ефоре\_фирст\_рекуест кука, 20 ефоре\_рекуест  
кука, 20, 122 ефоре\_рекуест руковаца са  
аутентификацијом,

208

Блеаџ пакет, 164 лока  
(у основним ша лонима), 29 доступних  
локова у Флакс-Боотстрап-у, 32 постова на логу,  
151-169 уредник за, 167-169 на страницама профила,  
154 дугачке листе са пагинирањем, 155-161 сталне  
везе ка, 165 -167 упита праћених постова помоћу  
придруживања ази података, 181-183 постова  
о огађеног текста са Маркдун и Флакс

Пагедовн, 161-165  
приказује пропраћене постове на почетној  
страници, 183-187  
подношење и приказ, 151-154 нацрта, 90-92  
криирање, 90 криирање нацрта аутентификације, 105  
криирање за РЕСТful АПИ, 203 о рађивача  
грешака, 91 регистрација са апликацијом у  
фа ричко функцији, 91 регистрација РЕСТful  
АПИ нацрта, 204 руте у, 91 БоолеанФиелд класа,  
109 оот.сх (скрипта за покретање контејнера), 259

Боотстрап, интеграција помоћу Флакс-Боотстрап-а,  
30-33

пословна логика, 25

## Ц

кардиналност, 58  
клик пакет, 1 о лак,  
имплементација до, 243 извештаја  
о покрivenости кода, 221-224 примера  
кода из ове књиге, 255 колекције (НоСКЛ  
азе података, 58 колона ( аза података),  
57 д .Цолумн класа, 62 СКЛАлцхеми колоне  
опције, 63 СКЛПалцхеми типа колона, 63  
интерфејс командне линије (ЦЛИ),  
инсталација за Хероку, 245 опција командне  
линије, флакс команда, 15 коментара (корисник) (погледајте  
коментаре корисника) урезивање сесија азе  
података, 67 условних изјава (у ша лонима), 28 , 48  
Цонфиг класа, 87 конфигурационих апликација креираних  
помоћу скрипте апликације, 93 апликације креираних  
фа ричком функцијом, 89 апликација постављених на  
Хероку платформи,

247

конфигурисање е-поште за апликације на Хер-оку, 248

Доџкер имплементација, 258 за  
споро извештавање о упитима, 238  
великих апликација, опције за, 86-88 слање е-  
поште за грешке у апликацији, 242  
Уник- азирани сервери, евидентирање,  
272 контејнера, 244, 256

(види и Доџкер)

преговарање садржаја, 205  
процесора контекста, 134  
контекста, 17 за е-пошту на  
позадинској нити, 83 контекст школке,  
додавање, 72 контролне структуре (Јиња2), 28  
колачића на страни клијента, корисничке сесије у,  
52 подешавања у о јекту одговора , 21 приказује  
пропраћене постове, 184

Координисано универзално време (УТЦ), 38 алатка  
за покрivenost, 222 (погледајте и извештаје о  
покрivenosti кода)

преате\_апп фа ричка функција  
која прилаже нацрт аутентификације апликацији,  
**106**

Напади фалсификовања захтева на више локација (ЦСРФ), **44**  
ЦСС

Боотстреп ЦСС датотеке, **30, 33**  
Боотстреп класе пагинације, **159** класа за  
аватар на страници профил, **147** стилова за  
лог постове, **153** цУРЛ, **217** променљива  
текуће\_време, **39** променљива контекста  
цуррент\_сер, **113, 142** цуррент\_сер.цан функција,  
**132** функција цуррент\_сер.цан, **132** функција  
цуррент\_администратор, **132**  
цуррент\_сер.ис\_аутентицитет својство, **110,**  
**114**  
цуррент\_сер.\_гет\_цуррент\_о јеџт метода, **152** Цигвин, **2**

Д  
аза података (у УРЛ адресама азе  
података), **61** аза података, **57-77** креирање  
та ела, **66** рисање редова, **68**

Флакс подршка за, **ки**  
уметање редова, **66**  
интеграција са Питхон школком, **72**  
велике апликације које користе различите азе података,  
**87**  
евидентирање спорих перформанси, **237-239**  
прављење корисника својим след еницима у  
ази података, **186** управљање са Флакс-  
СКЛАлхеми, **61** миграција са Флакс-Миграте, **73-77**  
додавање више миграција, **76** надоградња азе  
података, **75** дефиниција модела, **62-64**  
модификација редови, **68**

НоСКЛ, **58**  
корисничких улога, **127-131**  
додавање нових улога у сесији љуске, **134**  
Модел поста за постове на логу, **151**  
Руковање текстом Маркдовн, **164**  
о ез еђивање азе података на Хероку, **246**  
Питхон оквири азе података, **59-60** упити  
праћених постова помоћу придрживања ази података,  
**181-183**  
фильтр\_ и филтер упита, **182**  
придружи филтер упита, **182**

упити редова, **68**  
релациони модел, **57**  
релација у, **64-65, 171-178** та ела  
асоцијација, **172** представљање  
коментара на лог постовима, **189-191**

подешавање у великој апликацији, **96**  
СКЛ, **57** СКЛ наспрам НоСКЛ, **59** чување  
МДБ хешева за аватаре корисника, **148**  
употре а у функцијама приказа, **71-72** корисничких  
информација у, **137** корисника, хешева лозинки  
ускладиштењи у, **102** коришћење екстерне азе  
података са Доцкер контејнери, **264**

ДАТАБАСЕ\_УРЛ променљива окружења, **246,**  
**265**  
Валидатор поља ДатаРекуиред, **45, 49, 109** датума и  
времена, датум последње посете за кориснике, **138**  
локализација са Флакс-Моментом, **38** временских  
ознака у ази података о корисницима,  
**137**

Д о јекат, **62**  
Д .Цолумн класа, **62**  
Д .преате\_алл функција, **66, 75**  
Д .ФореигнКеј функција, **64**  
Д .релатионсхип функција, **64**  
Д .сесион о јекат, **67** Д .сесион.адд  
метод, **68** Д .сесион.цоммит метод ,  
**67, 121** Д .сесион.делете метода, **68**  
Д .сесион.ролл ацк метода, **68** режим  
отклањања грешака, **13, 93** --де уггер и --но-  
де уггер опије командне линије, **16** грешака  
током производње, **242** подсистема за  
отклањање грешака, **1** декоратор, **8** прилагођених,  
провера корисничких дозвола, **132** редослед, у  
приказу функција које користе више декоратора, **133**

закачице захтева имплементиране као, **20**  
ДЕЛЕТЕ метода захтева (ХТТП), **201** денормализација  
(НоСКЛ азе података), **59** зависности, **1** за  
апликације постављене на Хероку, **248** за развоју у  
односу на производњу, **156**

- инсталирање у виртуелно окружење са пип-ом, 5
- датотека са захтевима за велике апликације, 93
- команда распоређивања, 241
- распоређивање, 241-273
- Доцкер контејнери, 256-270 у о лаку, 243 грешке у евидентирању током производње, 242 на Хероку платформи, 244-255 припрема апликације, 244-253 традиционално, 270-273
  - увоз варија ли окружења, 271 подешавање сервера, 271 подешавање евидентирања, 272 ток после, 241 десериализација (АПИ), 212 развојни ве сервер, 10
- ДЕВ\_ДАТАБАСЕ\_УРЛ** променљива окружења, 96
- Доцкер, 244, 256-270
- прављење Доцкер слике, 257
  - оркестрација контејнера са Цомпосе, 265-269
- Доцкерфиле, 257
- слика команда, 260, 269 инсталација, 256 наред а за пријаву, 263 наред а евидентије, 262, 264 пс команда, 269 пусх команда, 263 рм команда, 262, 269 рми команда, 269 наред а за покретање, 262, наред а за заустављање
- Poj, 270
- системска команда, 269
- команда ознаке, 263
- коришћење екстерне азе података, 264
- коришћење у продукцији, 270 команда верзије, 256 команда јачине звука, 269
- Доцкер Цомпосе, 265 доцкер-
- компосе.имл датотека, 266 наред и дневника, 268 пс команда, 269 уп команда, 268 Доцкер Ху , 263 апликације и услуге укључене, 265
- документно оријентисане азе података, 57
- динамичке руте, 9
- динамичких УРЛ-ова, генерирање са функцијом урл\_фор, 37
- динос (Хероку), 244
- E**
- ЕЦ2 услуга (ABC), 243 е- пошта, 79 (погледајте и Флакс-Майл)
- конфигурисање за апликације на Хероку-у, 248 конфигурисање за Доцкер контејнер, 261 потврда корисничких налога, 120-124
  - руковање променама адреса за корисничке налоге, 125
  - слење за грешке у апликацији, 242
  - дијаграми односа ентитета, 58 .енв
  - датотека, 253, 266-267, 271 променљиве окружења дефинисане у Доцкерфиле-у, 258 увоз у традиционалној примени, 271 у конфигурацији велике апликације, 87
- Валидатор поља форме ЕкуалТо, руковање грешкама 116
- АПИ о рађивач грешака за ВалидационError, 212
  - о рађивача грешака у најчртима апликација, 91
  - О рађивач грешака Флакс-ХТТПАутх, 208 у РЕСТful ве услугама, 204
  - са преговарањем садржаја, 205
  - страница са грешкама, прилагођено, 33-36
  - грешака, евидентирање током производње, 242
  - проширења директиве, 30, 32 проширивост Флакс-а, ки екстензије, 1, 23, 30 додатних Флакс екстензија и пакета,
- 276
- иницијализација, 31
- Ф фа ричка функција, креирање апликација са, 88 лажних података лог постова, креирање, 155-157
- Факер пакет, 155 филтера ( аза података), 27 оффсет()
- филтер упита, 157 упита
  - праћених постова помоћу придрживања ази података, 183
- коришћење са упитима азе података, 68 филтер\_ и метод, 69
- СКЛАЦХеми филтери за, 69 флеш функција, 53

Фа ричка функција Флакс апликације, [88](#)  
 апп\_эррорхандлер декоратор, [91, 205](#) основна структура апликације са више датотека, [85](#)  
 прилагођених команди, [96](#) динамичких пута, [9](#)  
 екстензија, [30](#)

Флакс класа, [7](#)  
 флакс командних опција, [15](#)  
 инсталирање у виртуелно окружење са пип-ом, [5](#)  
 гашење сервера, [230](#) тест клијент, 224-229 рад са, додатни ресурси за, 275-277

флакс д команда довнграде, [76](#) флакс д миграте команда, 75-76 флакс д стамп команда, [76](#) флакс д команда надоградње, 75-76 флакс деплои команда, [268](#) Флакс Ектензион Регистри, [276](#) флакс рун команда, [10](#) --хост аргумент, [16](#) опција, [16](#) флакс скелл наред а, [15, 66, 72](#) флакс тест команда, --цовораге опција, [222](#)  
 Флакс-Боотстрап, 30-33 локова, [29](#)  
 иницијализација, [30](#) инсталирање, [30](#)  
 Куицк\_форм макро, [47](#) втф.куицк\_форм Јиња2 мацро, [110](#). Куицк\_форм метода, [47](#) Флакс-ХТТПАутх, [206](#)

о рађивач ефоре\_рекуест са аутентификацијом, [208](#)  
 о рађивач грешака, [208](#) иницијализација, [207](#) подршка за аутентификацију засновану на токенима, [209](#) Флакс -Логин, 107-115 додање  
 о расца за пријаву, [109](#) класа АнонимоусУсерМикин, [132](#) променљива контекста цуррент\_усер, [113, 152](#) како функционише, [1](#)Манагер\_манагер\_логин1 атри јут анонимоус\_усер, [132](#)  
 логин\_рекуириед декоратор, [108, 114](#)  
 функција логин\_усер, [111](#)

функција логоут\_усер, [113](#) припрема корисничког модела за пријављивање, [107](#)  
 пријављених корисника, [111](#) корисника који се одјављују, [112](#) тестирања пријављивања, [114](#)

УсерМикин класа, [107](#)  
 декоратор усер\_поадер, [108](#)

Флакс-Майл, 79-84  
 иницијализација, [80](#) интеграција е-поште са апликацијом, [81](#) слање асинхроне е-поште, [83](#) слање е-поште из Питхон скелл-а, [81](#) слање е-поште преко Гmail-а, [80](#)

Кључеви за конфигурацију СМТП сервера, [79](#)

Флакс-Миграте, 73-77  
 додавање више миграција, [76](#)  
 разматрања у промени шема азе података, [74](#) креирање или надоградња та ела азе података у великој апликацији, [96](#) иницијализација, [73](#) надоградња азе података, [75](#)

Флакс-Момент, [38](#)  
 функција формата, [40](#)  
 функција фромНов, [40](#)  
 функција локализације, [41](#)

Флакс-Пагедовн, 162-164  
 иницијализација, [162](#)  
 О разац за о јаву са омогућеним Маркдован, [162](#)

Флакс-СКЛАДЦХЕМИ, [60](#)  
 метода додања сесије, [67-68](#)  
 опција колона, [63](#) метода цреате\_алл, [66, 95](#) управљање азом података са, [61](#) метода рисања сесије, [68](#) метода дроп\_алл, [66](#) омогућавање снимања статистике упита, [239](#) филтер\_ и филтер за упит, [701](#) фирст\_куери\_4 метода, [139](#) функција гет\_де уг\_куериес, [237](#) функција гет\_оп\_404 погодности, [145](#) модела, [62](#)

МисКЛ конфигурација, [61](#)  
 метода пагинације, [158](#)  
 атри ута о јекта пагинације, [158](#)  
 метода о јекта пагинације, [159](#)  
 Постгрес конфигурација, [61](#)  
 извршилац упита, [69](#) филтера упита, [69](#)

о јекат упита ( азе података), 68  
 статистика упита које је снимио, 238 упита  
 за праћене постове помоћу придрживања азе података,  
 182

СКЛАЛЦХЕМИ\_ДАТАБАСЕ\_УРИ конфигурација, 61

СКЛАЛЦХЕМИ\_ТРАЦК\_МОДИФИЦА-  
 ТИОНС конфигурација, 61

СКЛите конфигурација, 61

Фласк-ССЛифи, 249 Фласк-ВТФ, 43

класа BooleanField, 109

конфигурација, 44

фалсификовање захтева на

више локација (ЦСРФ), 44 ДатаРекуиред

валидатор, 45, 109 онемогућавање ЦСРФ токена  
 у јединичним тестовима, 226 Фласк4Форм класа  
 форме поља, 45 Валидатор дужине, 109 формулаар  
 за пријаву, 109 класа ПасвордФиелд, 109

рендеровање, 47 класа СтингФиелд, 45, 109 класа

Су митФиелд, 45, 109 коришћење за приказивање  
 о расца, 47 валидате\_он\_су мит метод, 49,

валидатор Фласк 4, 111 класа Фласки, слика Доцкер  
 контејнера за, 260 фласки.пи, 266, 272 команда  
 покривености, 222 команда распоређивања, 241,

253, 255, 260 команда профила, 239

ФЛАСКИ\_АДМИН променљива окружења, 83,  
 131

ФЛАСКИ\_ЦОММЕНТС\_ПЕР\_ПАГЕ конфигурациона  
 променљива, 192

ФЛАСКИ\_ПОСТС\_ПЕР\_ПАГЕ конфигурациона променљива,  
 158

ФЛАСК\_АПП променљива окружења, 10, 66, 93,  
 246

подразумевано подешавање, 97

ФЛАСК\_ЦОНФИГ променљива окружења, 93,  
 247

ФЛАСК\_ЦОВЕРАГЕ променљива окружења,  
 223

ФЛАСК\_ДЕБУГ променљива окружења, 14, 93  
 подразумевано подешавање,  
 97 пратилаца, 171-187

односи азе података и, 171-178 на страници  
 профила, 178-180 показујући праћене постове  
 на почетној страници, 183-187

за петље, 28

страних кључева, 57, 64

форм.хидден\_таг елемент, 47

форм.валидате\_он\_су мит метода, 142 функција

формата, 40 о разаца (погледајте ве о расце)

пакет фунџтоолс, 133

## Г

г контекстна променљива, 18, 21

ГЕТ метод захтева (ХТТП), 19 у РЕСТфул АПИ-  
 преусмеравањима, 51 у ХЕДМЕТ апликацијама, 201  
 руковалац ресурсима за  
 постове на логу, 213 функција прегледа која  
 рукује ГЕТ захтевима, 48 функција гет\_фласхед\_мессажес,

54

Гит

преузимање примера кода, 2 сервера

посвећена Хероку апликацији, 246 изврни код за

примере кода, 2 иницијализација апликација на Хероку  
 сервер помоћу гит пуш-а, 254

коришћење са апликацијама на Хероку, 244

Гмаил, Фласк-Майл конфигурација за, 80

Граватар сервис, 146

Гунициорн ве сервер, 251-252

## Х

хешеви

МД5 хеш за УРЛ адресе аватара, 146

кеширање, 148 хеширање лозинке,

102 користећи Веркзеуг сигурносни

модул, 102

ХЕАД метода захтева (ХТТП), 19 помоћ од

Фласк заједница програмера, 276

Хероку платформа, 244-255

додањање Процфиле-а, 252

додањање датотеке са захтевима највишег нивоа,

248 аддонс: команда креирања, 246

ЦЛИ алат, 245

ционфиг команда, 246

ционфиг:сет команда, 247

конфигурисање е-поште, 248

конфигурисање евидентирања,

247 наред а креирање, 246

креирање Хероку налога, 245 креирање апликације, 245 постављање надградње апликације на, 255 постављање апликација на, користећи гит пусх, 254 омогућавање ез едног XTTП-а са Флакс-ССЛифи, 249 команда за пријаву, 245 наред а за евидентију, 247 наред а за одржавање, 255 о ез јејување азе података, 246 преглед евидентије апликација, 255 покретање производног ве сервера, 251 тестирање са хероку локалном командом, 253 име хоста (УРЛ-ови азе података), 61 XТМЛ Маркдун-то-XТМЛ претварач, 164 приказивање о разацу у, 47-48 XTTП ( ез едно), омогућавање са Флакс-ССЛифи, 249 XTTП аутентификација, 206 XTTП методе, 19 и руковоащи ресурсима за РЕСТфул ве сервер-вице, 215 методе захтева у РЕСТфул АПИ-јима, 201 XTTП статусни кодови, 21 404 грешка, 33, 145 РЕСТфул АПИ о рађивач грешака за статус 403 код, 206 који враћају РЕСТфул АПИ-ји, 204 XTTПБасицДутх класа, 207 XTTПие, коришћење за тестирање ве услуга, 217-218

И слике (контейнер), 244 прављење слике доцкер контейнера, 257 чишћење, 269 увоз датотека, макро ша лона, 29 укључених датотека, 29 наслеђивање у Јиња2 ша лонима, 29, 31 метода инит\_апп, 88 метода инсерт\_ролес, 131 интегрисано развојно окружење (ИДЕ), 275 његов опасан пакет, 119 генерисање токена за потврду за корисничке налоге, 119 подршка за аутентификацију засновану на токенима, 209

Ј јаваСкрипт датотеке (Боотстррап), 30, 33 јаваСкрипт, момент.јс и лиотека, 38 Јиња2 пакет, 1, 26-30 лок директиве, 29 контролних структура, 28 за директиву, 28 макро директиве, 29 директиве проширења, 30 директиве увоза, 29, 47 укључује директиве, 29 ша лона за приказивање, 26 сигурних филтера, 28 постављених директиве, 195 супер макроа, 30 варија ли, 27 филтера за, 27 втф.куицк\_форм макроа, 110 спојева ( аза података) спојени за претходне референце, 175 користећи у ази података упит праћених постова, 181 јКуери.јс и лиотека, 39 JSON серијализовани ресурси до и од, 210-213 употреба у РЕСТфул ве услугама, 202 JSON ве потписи (JВС), 119 помоћних функција јсонифи, 203 спојне та еле (погледајте та еле асоцијација (подаци аза))

К аза података кључ/вредност, 57 Ку јрнетес, 270

Л кодови језика, 41 веза, 36 уредника лог постова, 168 умерених коментара у траци за навигацију, 193 сталне везе до постова на логу, 165-167 линк за уређивање профила, 142 линк за уређивање профила за администратора, 145 до коментара на лог постовима, 192 до странице корисничког профила у траци за навигацију, 140 функција локализације, 41

локализација датума и времена, 38  
 конфигурисање евиденције на Хероку  
     платформи, 247 Доцкер конфигурација за, 258  
     доцкер логс команда, 262 грешака током  
         производње, 242 прегледа дневника  
     апликација на Хероку платформи, 255  
     подешавање у традиционалним применама, 272  
     функција прегледа за пријаву, 111  
     логин\_манагер.анонимоус\_сервис атријут, 132  
 логин\_манагер.сервис\_лоадер декоратор, 108 логин\_рекуирејт  
     декоратор, 108, 114, 122, 208

**M**

макрои, 29  
 МАИЛ\_ПАССВОРД променљива окружења, 80  
 МАИЛ\_УСЕРНАМЕ варијајла окружења, 80 маке\_респонсе  
     функција, 21, 185 релација „више према много“, 65, 172  
     напредна, 174 помоћне методе следеника, 176  
         имплементација као релације „један-према-више“,  
         175 самореференцијална, 174 „више-према-  
             више“ једна веза, 65

**Маркдовић , 162-165**

конверзија у ХТМЛ на серверу, 164 пост форма  
     омогућена за, 162 поруке, флешовање из  
 о разаца, аргумент 53-55 метода, 48 микросервиса,  
     265

Мицрософт Виндовс (погледајте Виндовс системи)  
     скрипте за миграцију (аза података), 74  
         криирање помоћу фласк д миграте, 75  
     дефиниција модела (аза података), 62-64  
         и лиотека момент.јс, 38  
         опције форматирања датума и времена, 40  
 МиСКЛ азе података, 264-265, 271

**Н**

НамеФорм класа, 45  
 именских простора у нацртима, 92  
 НоСКЛ азе података, 57-58 Флак  
     подршка за, ки СКЛ азе  
     података вс., 59

**О**

ОАутх2 аутентикација, 80 мапера  
 о јеката-документата (ОДМ), 60 о јектно-  
     релационих мапера (OPM), 60 модела, 62 односа  
         један-према-више, 58, 64  
     та ела коментара за та елу корисника,  
     189 релација много-према-више имплементирана  
         као, 175 упити, 70 релације један-на-један, 65

ОПТИОНС метод захтева (ХТТП), 19 оркестрација  
 (контролер) са доцкер-ом  
     саставити, 265-269

**П**

Би лиотека Пагедовић, 162  
 пагинација великих  
     колекција ресурса, 216 дугих листа лог  
     постова, 155-161 додавање виџета за  
         пагинацију, 158-161 креирање лажних  
         података лог постова, 155 приказивање на  
         страницама, 157 коментара корисника на  
     постове, 192  
 ПассвордФијелд класа, 109  
 лозинки, лозинка у УРЛ  
     адресама азе података, 61  
         ез једност, 102-105 ажурирања и  
         ресетовања корисничких налога, 125 перформанси,  
     237-240  
         евидентирање спорих перформанси азе података,  
     237-239 профилисање изврног кода, 239-240 дозвола,  
 127 модерирања коментара, 193 проверавања прилагођених  
     декоратора, 132 процена за корисника, 132 у ази  
     података корисничких улога, 128 константи за дозволе,  
     128 метода за управљање, 129 улога подржано са  
     дозволама, 130 јединичних тестова за, 134  
     пермисион\_рекуирејт декоратор, 214 пип, 5

Платформа као услуга (Паас), 244

Метод ПОСТ захтева (ХТТП), 43 о РЕСТфул  
     АПИ-јима, 201 одговор за преусмеравање  
     на ПОСТ захтеве, 51 о рађивач ресурса за  
     постове на логу, 214 функција прегледа која  
     рукује ПОСТ захтевима, 48

Пост/Редиреџт/Гет паттерн, 52 логин  
анд, 112  
Постфикс, 271  
Постгрес азе података, 246, 271  
логика презентације, 25 примарни  
кључ ( аза података), 57 колона  
примарног кључа, Фласк-СКЛАЦХеми  
услов за, 64  
Профил (апликације на Хероку), 252 ЦСС класа  
заглавља профила, 147 ЦСС класа са сличицама  
профила, 147 профила, 137-149 корисничко име  
и аватар аутора лога, линкови до странице  
профила, 153 постова на логу на страницама  
профила, 154 креирање корисничког профила стр,  
138-141 уредник за, 141-145

урдник на нивоу администратора, 143-145  
урдник на нивоу корисника, 141 пратилац  
на страници профила, 178-180 информација у,  
137 корисничких аватара, 146-149 изврни код  
за профилисање, 239 прокси сервера, 250, 271  
псицплг2 пакет, 248

Метод ПУТ захтева (ХТТП), 201  
PUT о рађивач ресурса за постове на логу, 215  
пимскл пакет, 264  
Питхон, 1  
kreирање виртуелних окружења са Питхоном  
2, 3  
kreирање виртуелних окружења са Питхоном  
3, 3  
оквира азе података, 59-60  
интеграција азе података са Питхон школјком, 72  
инсталирања пакета са пип-ом, 5 слика  
интерпретатора на Доцкер Ху -у, 258  
Индекс пакета Питхон, 276  
питхон-дотенв пакет, 272

К  
о јекат упита ( аза података), 68

Р  
функција преусмеравања, 22, 53  
преусмеравања, 22, 51-53  
ССЛ, 249  
Валидатор поља о расца Регекп, 116  
регистрација нових корисника, 115-118

додавање о расца за регистрацију корисника,  
115 регистрованих корисника, 117 регресија,  
221 релациона аза података, 57  
(погледајте и СКЛ азе података)  
Фласк подршка за, 21  
релације ( аза података), 57, 64-65, 171-178 динамичке  
везе, 70 много-према-више, 172 напредно, 174  
испитивање односа један-према-више, 70  
самореференцијално, 174

СКЛАЦХеми опције за, 65  
релативне УРЛ адресе (у везама), 37  
РЕМЕМБЕР\_ЦООКИЕ\_ДУРАТИОН опција,  
111  
протоколи позива удаљене процедуре (РПЦ), 199  
рендеровање (ша лони), 25 функција рендер\_темплате,  
27, 49, 53  
Контекст захтева за трансфер стања (погледајте РЕСТ),  
18 циклус захтев-одговор, 17-22 контексти апликације  
и захтева, 17

Методе ХТТП захтева, 19 тела  
захтева и одговора у РЕСТфул-у  
АПИ-ји, 201  
отпремање захтева, 18  
закачивања захтева, 20  
метода захтева у РЕСТфул АПИ-јима, 201 о јекат  
захтева, 19 захтева, 8 формата одговора за  
РЕСТфул АПИ клијенте,  
205  
одговори, 8, 21  
о јекат одговора, 21  
датотека са захтевима, 93, 248  
датотека са захтевима за развој, 156 датотека  
са захтевима највишег нивоа за Хероку платформу,  
248  
ресурси  
имплементација крајњих тачака за, 213-216  
пагинација великих колекција, 216  
сервијализација у и из ЈСОН-а, 210-213  
УРЛ адресе за,  
202 ресурса за рад са Фласк-ом, 275-277 до ијање  
помоћи, 276 укључење у Фласк заједницу,  
277

интегрисана развојна окружења  
(ИДЕ), 275

РЕСТ, 199

дефинисања карактеристика за архитектуру ве  
сервиса, 199 тела захтева и одговора, 201

метода захтева, 201 ресурса, концепт, 200 верзија  
ве сервиса, 202

РЕСТфул ве услуге са Фласком, 203-218 креирање  
АПИ нацрта, 203 руковање грешкама, 204

имплементација крајњих тачака ресурса,  
213-216 пагинација великих колекција ресурса, 216

серијализација ресурса у и из ЈСОН-а, 210-213

тестирање са ХТТПије -ом,  
аутентификација заснована на токенима  
217-218 , аутентификација корисника 208-210 са Фласк-ХТТПАут,  
206

- о рутни прокси сервери, 250, 271
- огата интернет апликација (РИА), 199
- постова о огађеног текста, користећи Маркдован и Фласк  
ПагеДовн, 161-165
- руковање огатим текстом на серверу, 164
- улоге, 127-135 додавање развојној ази података  
у сесији љуске, 134 додељивање, 131 представљање  
азе података, 127-131 улога креирања метода,  
130 јединичних тестова за, 134 верификација, 132-135  
враћање сесија азе података, 68

Ронахер, Армин, 1

декоратор руте, 133

руте, 8

приступ само аутентификованим корисницима,  
108 генерисање токена за аутентификацију,  
210 подршка за коментаре на лог постовима,  
191 модерирање коментара, 196 динамичких,  
12 уређивача за постове на логу, 167 праћење  
руте, 179 пута почетне странице са о јавама  
на логу, 152 подршка за пагинацију, 157 у  
нацрту аутентификације , 105 у нацртима, 91

стална веза до постова на логу, 165

пута за уређивање профила,  
141 пута за уређивање профила за администраторе,  
144 пута странице профила, 138 пута странице  
профила са о јавама на логу, 154 пута за  
регистрацију са е-поруком за потврду,  
120

из оп свих или праћених постова, 184

одјава, 113 регистрација корисника, 117

подсистема за рутирање, 1 ред ( аза  
података), 57 рисање, 68 уметање, 66 измена,  
68 упит, 68

С

тајни кључ, 44, 87, 119  
за апликације на Хероку платформи, 247 сигуран  
ХТТП, 249

СелектФиелд класа, 144

Селен, тестирање од краја до краја са, 230-234

самореферентне везе, 174

Сендмайл, 271

серијализација

десеријализација ресурса из ЈСОН-а, 212

серијализација ресурса у ЈСОН, 210 подешавање  
сервера у традиционалној примени, 271 гашење  
сервера, 230 променљива контекста сесије, 18, 52

сесион.get метод, 53 сесије ( аза података), 67

урезивања, 67 враћања, 68 сесија (корисник), 44

преусмеравања и, 51 метод сет\_џоокие, 21, 185

процесор контекста љуске, 72 СМТП сервер, 79

смтпли пакет, 79 профилисање извornог кода,  
239 СКЛ (Структурд Куери Лангуаге), 57 СКЛ аза  
података, 57 НоСКЛ вс., СКЛАлчеми, 60 опција  
колона, 63 типа колона, 63 провера извornог СКЛ  
упита генерисаног од стране,

Маркдован конверзија текста у ХТМЛ, 164 извршиоца упита, 69 филтера упита, 69 опција односа, 65

СКЛАЛЦХЕМИ\_ДАТАБАСЕ\_УРИ, 61, 87  
СКЛАЛЦХЕМИ\_ТРАЦК\_МОДИФИЦАЦИОНС, 61  
СКЛите аза података, 271  
ССЛ\_РЕДИРЕЦТ променљива, 249, 250 трагова стека, 242 ве сервиса ез стања, 206 статичких датотека, 37 статичких метода, 131 статусних кодова (ХТТП), 21

СтрингФиелд класа, 45, 109  
Су митФиелд класа, 45, 109 супер функција, 30, 33 системски дневник, 272

т \_та ленаме\_ променљива класе, 62 та еле ( аза података), 57 креирање, 66 креирање или надоградња у великој апликацији, 96 спаја, 58

теардовн\_рекуест кука, 20 ша лона, 25-41 додавање класе дозволе у контекст ша лона, 134

Боотстрап интеграција помоћу Фласх-а

Боотстрап, 30-33

модерирање коментара, 194 е-

порука са потврдом коју користи нацрт аутентификације, 121 прилагођена страница са грешком, 33-36

дефинисано, 25 уређивање ша лона лог постова, 167 приказивања фласх порука, 54

Декларација Фласк-Пагедовн ша лона, 162 по ољшања пратиоца у заглављу профила, 178

за нацрт аутентификације, 105 за поруке е-поште, 81 за формулар за пријаву, 109 за о разац за регистрацију новог корисника, 116 за кориснички профил, 139 поздрав пријављеног корисника, 114 ша лон почетне странице са о јавама на логу, 152

Јинъа2 ша лонски механизам, 26-30 линкова, 36 локализованих датума и времена са Фласком Тренутак, 38 ша лона за пријаву, ажурирање да и се приказао о разац за пријаву, 112 пагинација подножја за листе постова на логу, 160 макро ша лона за пагинацију, 159 трајних веза до постова на логу, 166 ша лона странице профила са о јавама на логу, 155 статичних датотека, 37 фолдера са ша лонима, 26 коришћење да прикажете о разац у ХТМЛу, 47

тест команда за покретање јединичних тестова, 95 тестирање, 221-235 процена вредности, 234 енд-то-енд, користећи селен, 230-234 до ијање извештаја о покривености кода, 221-224 ве сервиса са ХТТПиес-ом, 217-218 тестови хеширања лозинке, 104 файл тестова јединица за велике апликације, 94 јединичних теста за улоге и дозволе, 134 користећи Флакс тест клијент, 224-229

тестирање ве апликација, 225-228 тестирање ве сервиса, 228-229 провера функционалности пријављивања, 114 временских (погледајте датуме и време) временских ознака, рад са, користећи Флакс Тренутак, 39 аутентификација заснована на токенима, 208-210, 218 трансакција, 67 (погледајте и сесије ( аза података))

у непотврђени рачуни

фильтрирање у о рађивачу ефоре\_апп\_рекуест, 122 страница на којој се тражи потврда налога, 123 униттест пакет, 95

УРЛ фрагменти, 192

УРЛ адресе

руте апликација, 9 УРЛ мапа апликације, 19 аватар, 146 аза података, у Флакс- СКЛАЛЦХЕМИ, 61 за ресурсе, 200 за ресурсе потпуно квалификован, 202

у имејловима за потврду за корисничке налоге, 121  
у везама, 37  
функција урл\_фор, 37, 53, 92, 121 аргумент  
урл\_префикс, 106 аутентификација корисника (погледајте аутентификацију) коментари корисника, 189-197 представљање аз е података, 189-191  
модерирање, 193-197 подношење и приказ-, 193 функција учитавања корисника, 108

## Модел корисника

припрема за пријављивање, 107  
припрема за хеширање лозинке, 102 генерисање токена корисничког налога и верификација- ција, 119  
корисничких профиле (погледајте профиле) корисничке улоге (погледајте улоге) корисничке сесије, 44, 52  
истека, дугорочни колачић за, 111 Усер.цан метод, 153 корисничко име (УРЛ-ови аз е података), 61  
корисника чинећи постојеће кориснике својим пратиоцима, 186  
стварање сопствених след еника на изградњи, 186

УТЦ (координисано универзално време), 38 уВСГИ ве сервер, 252

В метод валидате\_он\_су мит, 49, 111  
ВалидационЕррор, 116, 212  
РЕСТфул АПИ о рађивања грешака за, 213  
валидатора, 44, 109 утврђених, ВТФормс пакет, 46  
имплементираних као методе, 116 варија ли, 26-27 филтера за, 27 вен пакет (Питхон), 3  
верифи\_пассворд метода, 111 функција прегледа, 8 потврда кориснички налоги, 121 коришћење аз е података, 71-72 за коментаре на лог постовима, 191 за уредника лог постова, 168 за пратиоце на страници профила, 179 за трајне везе до постова на логу, 165 за рукување о расцима, 48-51

у нацрту аутентификације, 105 у нацртима, 92 имплементације функције за пријаву, 111 редоследа декоратора, 133 сврхе, 25 виртуелних окружења, 2 активирања, 4 креирања са Питхон-ом 2, 3 креирања са Питхон-ом 3, 3 деактивирања, 5 инсталирања Фласк-а у, 5 коришћење ез активирања, 5 виртуелних машина, 256 виртуелних сервера, 243 виртуален услугни програм, 3 волумена (Доцкер), уклањање, 269

## В

ве сервер за коно арице, 252  
ве претраживача, алатка за аутоматизацију Селениум, 230  
ве о расца (Хероку), 244 ве формулара, 43-55 о расца за логове, 151 конфигурација, 44 трептање поруке, 53-55 класе о разаца, 44-47 које генерише Фласх -ВТФ, ЦСРФ токени у,

## 226

руковање функцијама приказа, 48-51  
ХТМЛ приказ, 47-48 у нацртима, 92  
формулар за пријаву, 109 о разац за уређивање профила, 141  
о разац за уређивање профила за администраторе, 143 преусмеравања и корисничке сесије, 51-53 о разац за регистрацију корисника, 115

Интерфејс мрежног пролаза за ве сервер (ВСГИ), 1, 7 ве сервера који се инсталирају у традиционалној примени, 271 радни ве сервер на Хероку-у,

## 251

ве сервиси, 199, 203  
(погледајте РЕСТфул ве услуге са Фласком)  
тестирање РЕСТфул ве сервиса користећи Флак тест клијент, 228-229  
Веркзеуг ( ез једносни модул), 1, 102, 242  
генерате\_пассворд\_хасх функција, 103  
ПрофилерМидлеваре ВСГИ средњи софтвер, 240  
Прокиф ВСГИ средњи софтвер, 250

метода верифи_пассворд, 103	ВТФормс пакет, 43
Виндовс подсистем за Линук (ВСЛ), 2	урађени валидатори, 46
Виндовс системи, 2	Валидатор регуларних израза, 116
Доцкер за Виндовс, Хипер-В функција и, 257	СелеџФиелд класа омотача, 144
тестирање Хероку имплементације, коришћењем Бе сервера Ваитресс, 252 радни динос (Хероку), 244 втф.куицк_форм функција, 47	стандардна ХТМЛ поља подржана од, 45
	Икс
	КСМЛ у РЕСТфул ве сервисима, 202

## О аутору

---

Мигел Грин је првимајући аутор који има преко 25 година искуства као софтверски инжењер. Има [лог](#) где пише о разним темама, укључујући већи развој, ројотику, фотографију и повремене филмске прегледе. Живи у Портланду, Орегон.

## Колофон

---

Животиња на насловној страни Flask Be Девелопмент-а је пиренејски мастиф (раса Цаница лупус фамилиарис). Ови циновски шпански пси потичу од древног пса чувара стоке званог Молосус, којег су узгајали Грци и Римљани и који је сада изумро. Међутим, познато је да је овај предак играо улогу у стварању многих раса које су данас уочијене, као што су ротвајлер, немачка дога, њуфаундленд и кане корсо. Пиренејски мастифи су признати као чиста раса тек од 1977. године, а Амерички клуб пиренејских мастифа ради на промовисању ових паса као кућних љубимаца у Сједињеним Државама.

После Шпанског грађанског рата, популација пиренејских мастифа у њиховој родној домовини је опала, а раса је преживела само захваљујући преданом раду неколико раштрканих одгајивача широм земље. Савремени генетски фонд за Пиринејце потиче од ове послератне популације, што их чини склоним генетским болестима као што је дисплазија кука. Данас, одговорни власници се старају да њихови пси буду тестирани на олести и рендгенски дају и потражили ако нормалности кука пре него што буду узгајани.

Одрасли мужјаци пиренејског мастифа могу достићи и више од 200 фунти када су потпуно одрасли, тако да поседовање овог пса захтева посвећеност до роја је учи и пуно времена на отвореном. Упркос својој величини и историји ловаца на медведе и вукове, пиринејци имају веома миран темперамент и одличан су породични пас. На њих се може ослонити да ће бити ринути о деци и заштитити дом, док су у исто време послушни са другим псима. Уз одговарајућу социјализацију и снажно вођство, пиренејски мастиф напредује у кућном окружењу и је одличан чувар и пратилац.

Многе животиње на насловницама O'Reilly-а су угрожене; сви су они важни за свет. Да сте сазнали више о томе како можете да помогнете, посетите [animal.oreilly.com](#).

Насловна слика је из ЈГ Вуд'с Анимате Цреатион. Фонтови насловнице су УРВ Типеритер и Гуардиан Санс. Фонт текста је Адо је Минион Про; фонт наслова је Адо је Мириад Цонденсед; а фонт кода је Ујенту Моно Далтона Маага.