Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Sever

# Podpora izvajanju spletnih kvizov in preverjanj znanj

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana 2013

*Besedilo je oblikovano z urejevalnikom besedil LaTeX.*

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

*Tržaška 25*
*1001 Ljubljana, Slovenija*
*telefon: 01 476 84 11*
*01 476 83 87*
*faks: 01 426 46 47*
*01 476 87 11*
*www.fri.uni-lj.si*
*e-mail: dekanat@fri.uni-lj.si*

Št. naloge: 00080/2013
Datum: 08.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **NEJC SEVER**

Naslov: **PODPORA IZVAJANJU SPLETNIH KVIZOV IN PREVERJANJ ZNANJ**
**SUPPORT FOR PERFORMING ONLINE QUIZES AND EXAMINATIONS**

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

V zadnjem času je vedno bolj priljubljeno izobraževanje na daljavo in v okviru tega tudi preverjanje znanja. Na voljo imamo številne rešitve za izobraževanje na daljavo, kot sta Coursera, Moodle ipd. Vedno pa obstaja težava, kako pri preverjanju znanja zagotoviti, da si uporabnik ne pomaga s kakšnimi nedovoljenimi pristopi. V okviru diplomske naloge bi bilo potrebno pripraviti prototip sistema, ki na odjemalčevi strani s pomočjo JavaScript jezika preverja regularnosti na strani odjemalca in anomalije preko REST načina sporoča na strežniški del, kjer se izvaja analiza (npr. uporabnik je v zadnji minuti večkrat izgubil fokus nad osnovnim oknom brskalnika, poskušal je kopirati vsebino okna ipd.). V okviru rešitve naj bo podprto tudi spremljanje goljufij v živo s podporo učilnic in sedežnega reda. Ključna zahteva je, da rešitev deluje v odjemalčevem spletnem brskalniku, brez potrebe po nameščanju dodatne programske opreme.

Mentor:

doc. dr. Dejan Lavbič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

# Izjava o avtorstvu diplomskega dela

Spodaj podpisani Nejc Sever, z vpisno številko **63100347**, sem avtor diplomskega dela z naslovom:

*Podpora izvajanju spletnih kvizov in preverjanj znanj*

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Dejana Lavbiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 13. septembra 2013 Podpis avtorja:

*Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za pomoč, nasvete in odzivnost na zastavljena vprašanja pri izdelavi diplomske naloge.*

*Prav tako se zahvaljujem sestrama za podporo pri študiju in bratrancu za pomoč pri izdelavi diplomske naloge.*

*Največja zahvala gre mami za finančno in moralno podporo tekom študija.*

# Kazalo

# Kazalo slik

# Seznam kratic in simbolov

| | |
|---|---|
| **AJAX** | Asynchronous JavaScript and XML |
| **API** | Application programming interface |
| **CSS** | Cascading Style Sheets |
| **DAO** | Data access object |
| **DOM** | Document Object Model |
| **FIFO** | First In First Out |
| **HTML** | HyperText Markup Language |
| **HTTP** | Hypertext Transfer Protocol |
| **HTTPS** | Hypertext Transfer Protocol Secure |
| **IDE** | Integrated Development Environment |
| **IP** | Internet Protocol |
| **Java EE** | Java Enterprise Edition |
| **JPA** | Java Persistence API |
| **JPQL** | Java Persistence Query Language |
| **JSON** | JavaScript Object Notation |
| **JSP** | Java Server Pages |
| **LMS** | Learning Management System |
| **Moodle** | Modular Object-Oriented Dynamic Learning Environment |
| **OS** | Operacijski Sistem |
| **SDK** | Software Development Kit |
| **SQL** | Structured Query Language |
| **SSL** | Secure Socket Layer |
| **URL** | Uniform resource locator |

# Povzetek

Primarni cilj diplomske naloge je razviti delujoč prototip sistema za podporo pri reševanju spletnih kvizov na učnem sistemu Moodle, ki zaznava in do neke mere odpravlja goljufije. Te je potrebno zaznati na strani odjemalca, v našem primeru študenta, z uporabo skriptnega programskega jezika Javascript in jih posredovati strežniku za nadaljnjo obdelavo. Funkcionalnosti tehnologije Google App Engine so omogočile brezplačno delovanje prototipa z naprednimi elementi programskega jezika Java EE. Sistem omogoča dodajanje novih kvizov, urejanje sedežnega reda učilnic, spremljanje goljufij v živo in analizo zaznanih goljufij. Prototip je zasnovan in narejen tako, da deluje na večih operacijskih sistemih in spletnih brskalnikih. V diplomski nalogi so kot prve analizirane že obstoječe tehnologije, nato opisane uporabljene tehnologije za doseganje rezultatov, sledi implementacija sistema in sklepne ugotovitve. Projekt je odprtokoden in na voljo vsem, ki jih zanima izvorna koda rešitve.

**Ključne besede:**
Moodle, Google App Engine, Javascript, goljufanje, analiza, kviz

# Abstract

The primary goal of this thesis is the development of a working prototype of a support system for online quizzes in the Moodle Learning Management System. The system detects and to a certain degree prevents cheating on the quiz. The detection takes place on the client side, in our case on the student's computer, by using the JavaScript scripting language and is forwarded to the server side for further processing. The functionality of the Google App Engine enabled a free of charge deployment of a prototype that uses the Java programming language. The system allows for creating quizzes, managing classroom seating arrangements and enables both realtime cheating alerts and post-quiz analysis of cheating. The prototype is designed and implemented to support multiple operating systems and web browsers. In the thesis we first analyze existing technologies and describe the technologies used in the implementation. Next we describe the actual implementation of the prototype and finally we offer some closing thoughts. The project is open source and is freely available.

**Keywords:**
Moodle, Google App Engine, Javascript, cheating, analysis, quiz

# Poglavje 1

# Uvod

S hitrim razvojem spletnih tehnologij se širi tudi izobraževanje na daljavo. Čez čas se je razvilo nekaj spletnih mest (Coursera [1], Udacity [2]), ki ponujajo kvalitetno vsebino priznanih svetovnih fakultet. Oddaljeno izobraževanje lahko izvajamo na različne načine, od ogleda videoposnetkov lekcij do spletnih govorilnic in predstavitev. Težava se pojavi pri ocenjevanju pridobljenega znanja na daljavo, saj težko izvajamo nadzor nad uporabo nedovoljenih pristopov.

Pri ocenjevanju znanja posameznika se v večini primerov iz prakse uporablja vprašalnike v obliki spletnih kvizov ali manjše projekte v obliki domačih nalog. Pri oddaji domačih nalog je dovoljena uporaba določene literature, vendar se naloge ne sme prepisati. Za ugotavljanje avtorstva je razvitih nekaj dobrih sistemov za zaznavanje plagiatorstva (npr. Moss [3]). Pri reševanju spletnih kvizov je namen preverjati naučeno znanje, pri čemer ni dovoljena uporaba dodatnih pripomočkov. Problem pri reševanju spletnih kvizov je v zaznavi in odpravljanju goljufij. Spletni kvizi niso koristni le pri učenju na daljavo, temveč učiteljem tudi olajšajo delo in odpravljajo napake pri ocenjevanju ter omogočajo priročnejše arhiviranje rezultatov.

V diplomski nalogi bomo analizirali pogoste goljufije in razvili sistem za podporo pri reševanju spletnih kvizov na sistemu Moodle. Cilj je razvoj brezplačnega odprtokodnega sistema, ki omogoča zaznavo, odpravljanje in ana-

lizo goljufij. Na strani odjemalca bo za zaznavo goljufij uporabljal Javascript in zabeležene dogodke pošiljal na strežnik. Strežnik bo poganjal Google App Engine [4], ki nam do neke mere omogoča brezplačno gostovanje.

Motivacija za pisanje diplomske naloge je spoznavanje nove tehnologije in majhen prispevek k reševanju problema goljufanja na spletnih kvizih. Ker je projekt odprtokoden, lahko v prihodnosti k razvoju prispeva kdorkoli in pripomore k implementaciji boljšega sistema.

V poglavju 2 bomo analizirali že obstoječe rešitve na področju preprečevanja goljufij in opisali, v čem se naš sistem od njih razlikuje. Nato v poglavju 3 sledi opis vseh tehnologij, uporabljenih pri implementaciji sistema. Uporabo tehnologij in podroben opis implementacije, delovanja ter uporabe vseh funkcionalnosti sistema bomo opisali v poglavju 4. Za konec v poglavju 5 sledijo sklepne ugotovitve, opis težav pri razvoju sistema in možnosti za nadaljnji razvoj.

# Poglavje 2

# Obstoječe rešitve

V tem poglavju bomo preučili in opisali, kakšne rešitve za preprečevanje goljufij na spletnih kvizih že obstajajo. Funkcionalnosti rešitev bomo najprej analizirali in nato izpostavili njihove pomankljivosti oz. omejitve, ki jih naš sistem rešuje.

## 2.1 Pregled obstoječih rešitev

### 2.1.1 Sistemi za zaznavo plagiatorstva

Na trgu obstajajo številni programi, namenjeni zaznavi avtorstva del, kot so domače naloge, izvorne kode programov, pisni sestavki in podobno. Nekateri omogočajo zaznavo vseh pisnih nalog, drugi so specializirani na zaznavo plagiatorstva strukture programskih jezikov. Sistemi delujejo s pomočjo podatkovne baze lokalno ali preko spletnih virov. S temi podatki nato primerjajo podobnosti dokumentov, kot so uporaba enakih spremenljivk in programskih konstruktov, neposredno prepisovanje ali prevajanje besedila in mnogo več. Podrobnosti o avtorstvu nato posredujejo uporabniku, ki presodi ali je oddan izdelek plagiat ali ne [5].

Znan sistem, ki je specializiran za pregled avtorstva programske kode, je Moss. To je brezplačen program, ki podpira zaznavo plagiatorstva v veliko različnih programskih jezikih[3]. Na spletu so na voljo vtičniki, ki omogočajo

vgradnjo sistema Moss v sistem Moodle [6]. Ta nudi možnost oddaje programerskih projektov, katerih podobnosti lahko preverjamo z omenjenim vtičnikom. Za sistem Moodle obstajajo tudi drugi vtičniki, ki imajo podobne funkcionalnosti. Drugi primer sistema za prepoznavo plagiatorstva je iThenticate [7]. Ta za razliko od Mossa ni specializiran le za programske jezike in je plačljiv. Ima svojo podatkovno bazo oddanih izdelkov po vsem svetu. Na trgu je še veliko drugih brezplačnih in plačljivih sistemov, vsi pa imajo podobne funkcionalnosti. Razlika je le v algoritmih iskanja podatkov ter v količini dostopnih podatkov o avtorskih delih.

### 2.1.2 Onemogočanje dostopa do aplikacij

Sistemi za onemogočanje dostopa do aplikacij (ang. lockdown systems) onemogočajo preklapljanje med aplikacijami operacijskega sistema. Uporabljamo jih lahko na računalnikih v javnih prostorih (npr. knjižnica, železnica,...), da uporabnikom onemogočimo dostop do operacijskega sistema in za njih nerelevantnih sistemskih aplikacij [8].

Izpeljanka sistemov za onemogočanje dostopa do aplikacij so sistemi za zaklep brskalnikov, ki uporabnika omejujejo le na uporabo aplikacije brskalnika. Take sisteme se uporablja proti goljufanju pri reševanju spletnih kvizov. Za delovanje je potrebno program naložiti na računalnik, na katerem se bo izvajal spletni kviz. Z nalaganjem programa se strinjamo, da ima le-ta dostop do funkcij operacijskega sistema, preko katerih uporabniku računalnika onemogoča dostop do drugih aplikacij in nedovoljenih funkcionalnosti. Nekatere rešitve imajo v programu svoj brskalnik, ki ima omejene funkcionalnosti [9].

Na trgu je nekaj dobrih rešitev, ki omogočajo zaklep brskalnikov in so v večini plačljive. Primer plačljivega programa je Respondus LockDown Browser [10], ki ima možnost povezave z različnimi učnimi okolji, med njimi tudi Moodle. Brezplačna različico takšnega tipa programa ponuja podjetje WebAssign, ki se prav tako imenuje LockDown Browser [11] in deluje na podoben način.

## 2.2 Predlogi za izboljšave

V poglavju 2.1 smo opisali, kakšne rešitve za zaznavo goljufij in plagiatorstva že obstajajo. V nadaljevanju si bomo pogledali prednosti in slabosti obeh vrst sistemov in ju primerjali z našo rešitvijo.

Prototip sistema, razvitega kot rezultat diplomske naloge, združuje lastnosti tako sistemov za zaznavo plagiatorstva kot sistemov za onemogočanje dostopa do aplikacij oz. brskalnika. Plagiatorstvo se preverja šele po oddanem izdelku, kar lahko v našem sistemu na nek način izvajamo z analizo poskusov reševanja kvizov. Po drugi strani nam sistem za onemogočanje dostopa do aplikacij omogoča preprečevanje goljufij med samim reševanjem, kar pri našem sistemu izvajamo na odjemalčevem sistemu s pomočjo Javascripta. Pri preverjanju prepisovanja med posamezniki obstajajo odprte možnosti za nadgradnjo, te so opisane v poglavju 5.2. Z njimi bi se približali sistemom za zaznavo plagiarizma.

Naš sistem je bližje sistemom za onemogočanje dostopa do aplikacij, natančneje sistemom za zaklep brskalnikov. Slabost obstoječih rešitev je v potrebnem nalaganju odjemalskih programov na vse računalnike, na katerih hočemo reševati spletni kviz [9]. Za rešitev te slabosti smo namestitev odjemalca centralizirali s premestitvijo na strežniško stran. Tako odjemalski del ni del operacijskega sistema, temveč je v obliki datoteke Javascript del spletne strani, na kateri se izvaja kviz. Zaradi tega nam ni potrebno na strani odjemalca naložiti posebnega programa, vendar moramo v zakup vzeti pomankljivost Javascripta pri izvajanju funkcij operacijskega sistema. To nam onemogoča preprečevanje nekaterih akcij, kot sta na primer preklapljanje oken in zajem zaslona. Druga prednost našega sistema pred sistemi za zaklep brskalnikov je analiza reševanja kvizov. Namesto, da goljufije le preprečujemo, nudimo vpogled v časovnico goljufij vseh poskusov in zajem goljufij v živo. S tem uporabniku nudimo celosten pregled nad prepovedanimi akcijami, ki se jih je uporabnik posluževal.

# Poglavje 3

# Uporabljena orodja in tehnologije

## 3.1 Google App Engine

Sistem za zaznavanje goljufij smo se odločili postaviti na storitev v oblaku Google App Engine. Google App Engine je sistem, ki poganja spletne aplikacije na infrastrukturi podjetja Google. Največja prednost je v enostavni uporabi in nudenju dodatnih storitev za uporabnike. Ni nam potrebno postavljati aplikacijskih in podatkovnih strežnikov, izvorno kodo aplikacije le naložimo na Google App Engine in aplikacija je dostopna celemu svetu. Poleg enostavnega zagona so nam na voljo različne storitve, kot so Tasks Queue (FIFO vrsta za izvajanje operacij), Memcache (strežniški predpomnilnik), Datastore (sistem hranjenja podatkov), pošiljanje elektronske pošte in veliko drugih [4].

V primeru, da se spletna aplikacija začne širiti tako v smislu hranjenja velikih količin podatkov kot v številu zahtevkov iz strani odjemalcev, nam to ne povzroča nobenih težav. Za vse podatke in zahtevke skrbi Googlov porazdeljen sistem strežnikov, ki so zanesljivi in učinkoviti. Poleg same učinkovitosti je potrebno gledati na ceno storitev. Eden izmed ciljev diplomske naloge je, da sistem deluje brezplačno. Tudi to nam Google App pretežno omogoča,

saj so storitve do neke mere brezplačne, če pa te meje presežemo, plačamo toliko, kolikor smo jih presegli. Nalaganje aplikacije na Google App Engine pa je brezplačno [12].

Google App Engine omogoča pisanje aplikacij v različnih programskih jezikih. Za programske jezike Java, Python, PHP in Go nudi celotno izvajalno okolje vključno s podatkovno bazo in funkcionalnostmi, sprecifičnimi za posamezne jezike. Za implementacijo sistema za preprečevanje goljufij smo se odločili uporabiti programski jezik Java, saj ima na razpolago dobra razvojna orodja in je zanesljiv, hiter ter varen jezik [13].

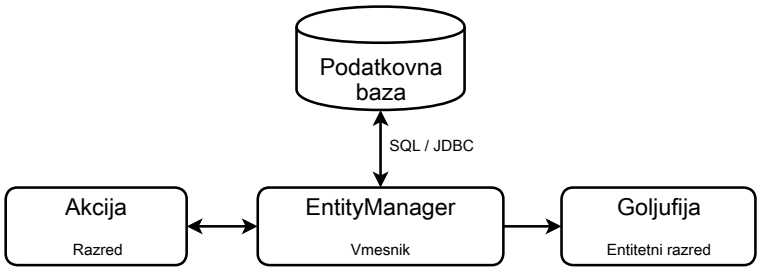
### 3.1.1 Izvajalno okolje Java

Za implementacijo sistema smo se odločili uporabiti programski jezik Java, za katerega nam Google App Engine nudi izvajalno okolje. Sem spadajo vmesniki za upravljanje z Java Servleti in podpora za storitve JPA, JavaMail in JCache [14]. Od naštetih tehnologij smo uporabili Java Servlete, JPA in nekaj aplikacijskih vmesnikov, ki jih bomo opisali v naslednjih poglavjih.

#### JPA

JPA je kratica za Java Persistence API, kar pomeni vmesnik za hranjenje trajnih podatkov. Trajni podatki so navadno hranjeni v entitetnih podatkovnih bazah, ki vsebujejo entitetne tabele in atribute. Tega v objektno usmerjenem programiranju ne poznamo, zato je potrebno trajne podatke iz entitet spremeniti v objekte, s katerimi lahko upravljamo v Javi. Ena izmed dobrih lastnosti JPA vmesnika, je neodvisnost od ponudnika podatkovne baze. Tako lahko vsako aplikacijo, napisano v Javi, prenesemo na podatkovno bazo drugega ponudnika, ne da bi morali posegati v izvorno kodo programa [15]. Ker uporablja Google App Engine za hranjenje podatkov BigTable, ki ni relacijska baza, je vmesnik JPA prilagojen in ima določene omejitve funkcionalnosti.

Vmesnik JPA preslika podatke iz podatkovne baze v javanske razrede. Razrede, za katere želimo, da so preslikani iz podatkovne baze ali obratno,




Slika 3.1: Vmesnik EntityManager

označimo z anotacijo `@Entity` in jih bomo od tega trenutka imenovali entitete oz. entitetni razredi. To označuje razred, katerega lastnosti trajno hranimo v podatkovni bazi. Entiteta predstavlja tabelo iz relacijske podatkovne baze. Vsaka entiteta ima atribute, ki so v relacijskih tabelah predstavljeni kot vrstice.

Še ena izmed dobrih lastnosti vmesnika JPA je poizvedovanje po entitetah in njihovih povezavah brez uporabe tujih ključev. Za to je odgovoren razred EntityManager, ki skrbi za konsistentnost podatkov med entitetami in podatkovno bazo in omogoča akcije branja, urejanja, brisanja in dodajanja entitet v podatkovno bazo. EntityManager je vmesnik med podatkovno bazo in entitetnimi razredi, preko katerega lahko drugi razredi manipulirajo s podatki. Slika 3.1 ponazarja, kako razred Akcija ureja podatke preko vmesnika EntityManager, ki skrbi za skladnost podatkov med entitetnim razredom in podatkovno bazo. Preko vmesnika lahko izvajamo poizvedovanje po podatkih, ki poteka v jeziku JPQL. Jezik je podoben jeziku SQL, glavna razlika je v tem, da operira nad javanskimi objekti in ne neposredno s tabelami podatkovne baze [15].

### JSP in Servleti

Za generiranje dinamičnih spletnih strani na strežniku smo uporabili tehnologiji Java Server Pages (JSP) in Servlete. JSP so dokumenti HTML, ki lahko vsebujejo javansko kodo. Koda se izvrši ob vsaki zahtevi in generira

statični HTML dokument, ki ga strežnik nato pošlje odjemalcu. Servleti pa so javanski razredi, v katerih se, prav tako kot v JSP, generirajo dinamične spletne strani. Vsi JSP dokumenti se ob izvajanju aplikacije prevedejo v Servlete, tako da so strani JSP dejansko tudi Servleti [16].

Tehnologiji nam omogočata arhitekturo modela, pogleda in kontrolerja. V primeru našega sistema so JSP strani pogledi, Servleti služijo za vodenje poslovne logike (kontroler), razredi DAO pa kot model, ki neposredno upravljajo s trajnimi podatki. Poizvedbe v naši implementaciji sistema pogosto vodijo do servletov, ki preko razredov DAO pridobijo zahtevane podatke in jih preko preusmeritve pošljejo stranem JSP, kjer se prikažejo. JSP strani nam tako služijo kot predloga za vizualizacijo podatkov, ki jih pridobimo od Servletov. Servleti so zadolženi za pridobitev podatkov preko razredov DAO, njihovo obdelavo (poslovna logika) in pošiljanje predlogi JSP.

### 3.1.2 Uporabljeni vmesniki

Za lažjo in hitrejšo implementacijo določenih funkcionalnosti sistema smo si pomagali z vmesniki, ki jih zagotavlja Google App Engine. V nadaljevanju bomo opisali vse vmesnike, ki smo jih uporabili pri razvoju rešitve.

#### Memcache Java API

Strežniško predpomnenje je dobro uporabiti pri nekaterih operacijah, ki zahtevajo vpogled v podatkovno bazo. Najpogosteje se uporablja za branje podatkov, ki so pogostokrat zahtevani s strani strežnika. Vmesnik za strežniško predpomnenje nam nudi Google App Engine in se imenuje Memcache Java API [17]. V njem hranimo podatke v obliki zgoščene tabele. Ključi tabele so znakovnega tipa String, vrednost tabele pa je lahko kateri koli objekt, ki implementira javanski vmesnik Serializable. Pri uporabi predpomnenja moramo biti pozorni na usklajenost podatkov med podatkovno bazo in predpomnilnikom, da morda ne pride do situacije odstopanja podatkov. Pozorni moramo biti tudi na dejstvo, da se lahko predpomnilnik izbriše kadarkoli.

Morebitno neusklajeno delovanje moramo pričakovati in preprečiti med programiranjem aplikacije.

### Task Queue Java API

Izvajanje operacij na strežniku izven obsega odjemalčeve zahteve je mogoče izvajati z vmesnikom Task Queue Java API [18]. Ta omogoča kreiranje FIFO vrste, katere elementi so operacije. Te operacije se nato izvajajo ena za drugo, kdaj se izvedejo pa je odvisno od tega, kakšno vrsto vmesnika imamo. Google App Engine omogoča dva tipa vrst. Prvi tip so vrste , ki izvajajo operacije z določeno hitrostjo, definirano v definiciji vrste (Push Queues), drugi tip pa so vrste, ki izvajajo operacije na zahtevo (Pull Queues). Pri implementaciji našega sistema smo uporabili vrste tipa Push Queues. Z njimi ob prejetih goljufijah le-te zapisujemo v podatkovno bazo s stopnjo ene goljufije na sekundo.

### Users Java API

Za vsak sistem, ki je namenjen le določenim uporabnikom, potrebujemo prijavne podatke uporabnika in overitev podatkov. Vmesnik Users Java API omogoča zaznavanje, če je uporabnik prijavljen v sistem z njegovim Google računom ali z računom iz lastnih Google domen [19]. Uporabnika lahko z uporabo vmesnika enostavno preusmerimo na prijavno stran ter generiramo povezavi za prijavo in odjavo. Na prijavni strani uporabnik vnese podatke o svojem računu, za overitev pa poskrbi vmesnik.

### Channel Java API

Vmesnik Channel Java API omogoča vzpostavitev trajne povezave med odjemalcem in strežnikom [20]. S pomočjo vmesnika lahko pošiljamo sporočila s strežnika, ne da bi moral za njih odjemalec pošiljati zahteve. Dober primer uporabe je zajem goljufij v živo, kjer ob prejeti goljufiji strežnik pošlje sporočilo vsem odjemalcem, ki spremljajo goljufije v živo. Povezavo med

odjemalcem in strežnikom določa žeton povezave. Postopek ustvarjanja povezave je sledeč: strežnik kreira kanal, po katerem bo pošiljal sporočila. Za kanal ustvari enolični ključ kanala, ki ga pošlje vsem Javascript odjemalcem. Odjemalci se nato preko ključa povežejo na kanal in poslušajo sporočila.

## 3.2 Eclipse IDE

Za hitrejše in preglednejše pisanje programske kode smo uporabili razvojno okolje Eclipse IDE verzije 4.3 Kepler. Eclipse je združenje posameznikov in organizacij, ki razvijajo brezplačno odprtokodno programsko opremo. Glavne funkcionalnosti razvojnega okolja so barvanje izvorne kode, avtomatsko dopolnjevanje med pisanjem ter razširljivost. Razširljivost omogoča velika količina vtičnikov, ki eclipse naredijo uporaben tudi v drugih področjih, ne le v hitrejšem pisanju izvorne kode. Za razvoj produkta diplomske naloge smo uporabljali različico Eclipse IDE for Java EE Developers, ki ima na voljo orodja za urejanje vseh javanskih tehnologij, pa tudi orodja za spletni razvoj v tehnologijah HTML, CSS, Javascript in ostalih [21].

Za verzioniranje izvorne kode smo uporabili GitHub vtičnik EGit[22]. Ta nam omogoča zagon ukazov operacijskega sistema ali uporabo grafičnega vmesnika, ki naredi verzioniranje hitro in enostavno. Na voljo so tudi orodja za ogled zgodovine sprememb in za reševanje konfliktov, do katerih lahko pride med razvojem izvorne kode projekta.

Delo smo si olajšali tudi z vtičnikom za razvoj aplikacij na Google App Engine, ki ga nudi podjetje Google [23]. Vtičnik omogoča kreiranje novih projektov, lokalni strežnik za zagon projekta ter enostavno nalaganje projekta na Googlov oblak, na katerem nudimo storitev uporabnikom. Najpomembnejša lastnost je možnost zagona projekta na lokalnem strežniku. Tu se vzpostavi lokalna podatkovna baza in vse storitve, ki tečejo na oblaku. To nam omogoča hiter razvoj in razhroščevanje projekta, obenem pa tudi simulacijo, kako bo projekt dejansko deloval na Google infrastrukturi.

## 3.3 Moodle

Sistem za zaznavo goljufij, ki smo ga razvili v diplomski nalogi, zaznava goljufije na spletnih kvizih sistema Moodle. To je odprtokodni sistem za organizacijo učenja na spletu (LMS). Omogoča enostavno kreiranje dinamičnih spletnih strani za centralizacijo podatkov in funkcionalnosti, namenjenih lažjemu učenju in sodelovanju. Sistem je potrebno naložiti na strežnik, nastaviti konfiguracijo, ki nam ustreza, in že lahko uporabniki kreirajo spletne strani. Na voljo je veliko orodij, kot so nalaganje snovi za učence, podpora pogovorov v forumih, skupinsko pisanje člankov wiki, reševanje domačih nalog in še veliko drugih [24]. Za našo diplomsko nalogo je najpomembnejše orodje reševanje spletnih kvizov.

Spletni kviz je seznam različnih tipov vprašanj, na katere študent odgovarja preko računalnika. Za pravilne odgovore dobi določeno število točk, ki na koncu tvorijo oceno znanja iz snovi, kateri je namenjen kviz. Na voljo je veliko tipov vprašanj, časovno omejeni in časovno neomejeni kvizi, kreiranje večih strani vprašanj, nastavljanje načina ocenjevanja in drugo. Ravno spletni kvizi sistema Moodle so orodje, pri katerem želimo zaznati in preprečiti goljufije v produktu te diplomske naloge.

## 3.4 Pomožne knjižnice

Za razvoj sistema smo si pomagali z že obstoječimi knjižnicami, ki nam olajšajo delo pri implementaciji, da se lažje usmerimo k doseganju ciljev končne rešitve. Uporabili smo knjižnice, ki pripomorejo k učinkovitejšemu programiranju ter take, ki pripomorejo k boljši vizualizaciji in hitrejši uporabi spletne strani.

### 3.4.1 jQuery

Najpomembnejša uporabljena knjižnica je knjižnica jQuery. jQuery je brezplačen odprtokoden projekt, ki programerju omogoča obsežen nabor funkcij,

napisanih v jeziku Javascript. Funkcije so namenjene enostavni manipulaciji dokumenta DOM, določanju akcij na dogodke, animiranju elementov spletne strani, asinhronim poizvedbam s strežnikom in mnogo več. JQuery omogoča implementacijo zmogljivih dinamičnih spletnih strani. Dve pomembni lastnosti knjižnice sta podprtost spletnih brskalnikov in razširljivost [25]. Podprtost brskalnikov si lahko razlagamo s tem, da funkcije knjižnice delujejo na večini obstoječih brskalnikov, kar programerjem prihrani veliko časa. Posledica tega je poenostavljeno pisanje vtičnikov, kar omogoča razširljivost.

### 3.4.2 Twitter bootstrap

Zagotavljanje preglednosti in enakomernega stila strani na vseh brskalnikih nam omogoča knjižnica Twitter bootstrap. To je odprtokodni projekt za lažji razvoj izgleda in upravljanja spletnih strani, ki je brezplačno dostopen na spletni storitvi GitHub [26]. Knjižnica vsebuje dokumenta CSS in Javascript, ki nam omogočata določanje tipografije, sistematično postavitev gradnikov v mrežo in izgled vnosnih form ter ostalih elementov. Pri projektu diplomske naloge smo uporabili knjižnico verzije 2.3.2. Na voljo je že nova verzija 3.0.0, vendar se je implementacija sistema nekoliko spremenila, zato naš sistem še vedno uporablja starejšo različico. Na spletni strani projekta je na voljo tudi urejevalnik teme, s katerim lahko poljubno priredimo izgled gradnikov spletne strani [27].

### 3.4.3 Google Charts

Drugi vizualizacijski vmesnik, ki smo ga uporabili je Google Charts. To je nabor knjižnic Javascript, ki omogoča pregledno vizualizacijo podatkov na spletnih straneh. Na voljo so mnoge predstavitve podatkov, od najrazličnejših grafov do predstavitve podatkov na zemljevidu sveta [28]. V diplomski nalogi smo pri vizualizaciji seznama poskusov uporabili vizualizacijo s pomočjo tabele. Ta nam poleg preglednega videza tabele omogoča še urejanje vrstic po izbranih stolpcih in razporeditev podatkov na več strani.

### 3.4.4 jQuery Timepicker

Zadnji vmesnik, ki omogoča hitrejšo in preglednejšo uporabo spletne strani, je okno za izbiro datuma in časa Timepicker. Namesto, da uporabnik ročno vpisuje datum in čas v vnosno polje, ki mora biti v pravilni obliki, se ob kliku na vnosno polje pojavi okno vmesnika. Okno vsebuje koledar za izbiro datuma ter drsnike, s katerimi nastavimo čas do minute natančno [29]. V diplomski nalogi se dva takšna gradnika pojavita pri filtriranju poskusov, opisanem v poglavju 4.7.1. Vmesnik temelji na knjižnici jQuery.

### 3.4.5 Simple JSON

Simple JSON je knjižnica, napisana v programskem jeziku Java. Java je objektno usmerjen jezik, zato pri programiranju upravljamo z objekti, ki imajo lastnosti shranjene v atributih in funkcionalnosti v metodah [13]. Knjižnica Simple JSON omogoča enostavno pretvarjanje tekstovnih spremenljivk, katerih vsebina je v formatu JSON, v javanske objekte in obratno. Tako lahko lažje upravljamo z asociativnimi seznami in objekti formata JSON [30]. Knjižnico smo pri diplomski nalogi uporabljali za lažji prenos in pretvarjanje parametrov med odjemalcem in strežnikom.

# Poglavje 4

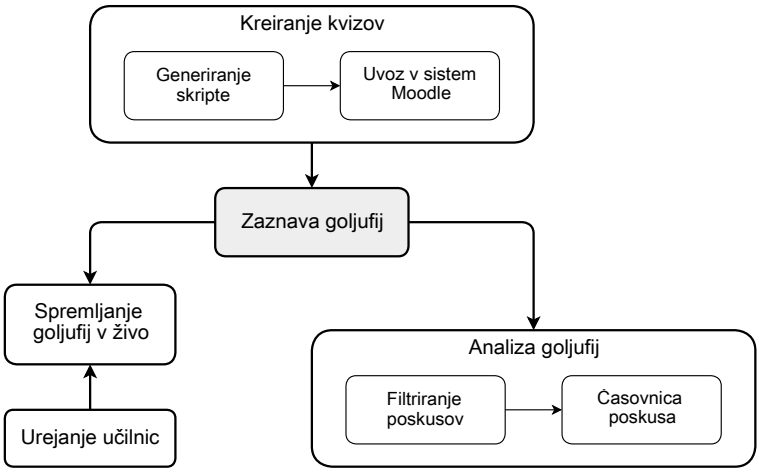
# Implementacija sistema

Rezultat diplomske naloge je programska rešitev, ki zaznava goljufije študentov ob reševanju spletnih kvizov v sistemu Moodle. Vsaka programska rešitev temelji na načrtovanju, implementaciji in testiranju določenih funkcionalnosti, ki ustvarjajo korist za končnega uporabnika. Ravno zaradi tega razloga je naslednje poglavje pomembno, saj opisuje ključne funkcionalnosti in implementacijo le-teh v razvitem sistemu. V podpoglavju 4.1 si bomo ogledali sistem kot celoto, v nadaljnjih pa ločene dele, ki omogočajo njegovo delovanje.

## 4.1 Celostni pogled na sistem

Za lažje razumevanje podrobnih opisov posameznih funkcionalnosti, ki bodo opisane v nadaljevanju poglavja, je dobro, da si predstavljamo sistem kot logično celoto z uporabo grafične vizualizacije. Na sliki 4.1 so v ta namen predstavljene ključne aktivnosti sistema.

V sistemu za zaznavo goljufij pri reševanju kvizov se ne moremo izogniti najpomembnejši aktivnosti, ki je sama zaznava goljufij. Pomembnost te aktivnosti je ponazorjena s tem, da je postavljena v središče samega grafa. Povezava med aktivnostima kreiranje kvizov in zaznava goljufij nam pove, da moramo za zaznavo imeti podatke o reševanem kvizu. Pred vsakim poskusom




Slika 4.1: Graf aktivnosti sistema

reševanja je zato potrebno kreiranje kviza, ki je sestavljeno iz faze generiranja skripte na implementiranem sistemu in faze uvoza omenjene skripte v sistem Moodle. Iz puščic, usmerjenih izven zaznave goljufij, razberemo, da gredo zaznane goljufije v aktivnosti analize goljufij in spremljanja goljufij v živo. Analiza goljufij tako kot kreiranje kvizov sestoji iz dveh faz. Prva faza je filtriranje poskusov, v kateri z različnimi filtri izločimo poskuse, ki jih želimo analizirati. V drugi fazi pa je možen ogled časovnice goljufij za izbrani poskus reševanja kviza. Spremljanje goljufij v živo je mogoče v učilnicah, katere zgradimo v aktivnosti urejanja učilnic.

## 4.2 Podatkovna baza

Sistem temelji na hranjenju podatkov, zato brez podatkovne baze ne more delovati. Preden se spustimo v podrobnosti samega entitetnega modela, na kratko opišimo, kako je sistem povezan s podatkovno bazo. Za hranjenje podatkov skrbi Google App Engine Datastore, za objektno-relacijsko preslikavo pa je zadolžen javanski vmesnik JPA. Struktura implementirane programske kode, ki skrbi za upravljanje s podatkovno bazo, je razdeljena na dva dela. Prvi del so JPA entitete, ki z javanskimi anotacijami [31] določajo strukturo

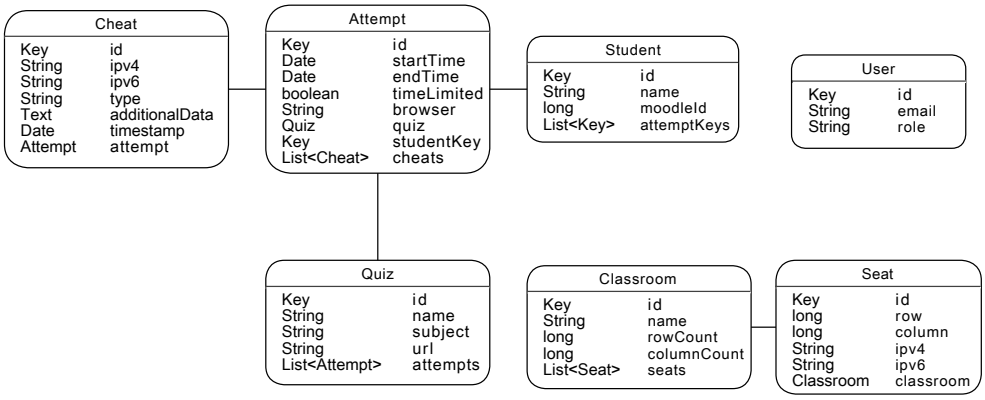
objektov, v katere se preslikajo podatki iz relacij. Drugi del pa so razredi DAO, ki skrbijo za osnovne operacije nad podatkovno bazo. To so operacije dodajanja, branja, urejanja in brisanja entitet. Za poizvedbe po entitetah JPA uporabljamo poizvedovalni jezik JPQL. Sintaksa tega jezika je zelo podobna jeziku SQL, razlikuje pa se v tem, da upravlja z entitetnimi objekti JPA, namesto neposredno z entitetnimi tabelami [15].

### 4.2.1 Struktura podatkovne baze

Za opis strukture podatkovne baze se bomo navezovali na entitetni model, ponazorjen na sliki 4.2. Na njem so predstavljeni razredi JPA, njihovi atributi in povezave med njimi. Na začetku omenimo, da ima vsaka entiteta enolični identifikator `id` tipa Key, da nam tega podatka ne bo potrebno ponavljati pri opisu vsake entitete.

Najprej si oglejmo najenostavnejšo entiteto, to je uporabnik (`User`). Načeloma so uporabniki predavatelji, ki uporabljajo sistem za preprečevanje in analizo goljufij. Atributa elektronski naslov (`email`) in vloga (`role`) sta znakovnega tipa String. S pomočjo elektronskega naslova in uporabo Users API vmesnika [32] ob prijavi uporabnika preverimo, če ima dostop do storitev sistema, sicer mu prijavo zavrnemo. Atribut vloga ima lahko vrednost "admin"(administrator) ali vrednost "user"(navadni uporabnik). Administrator lahko poleg ostalih storitev tudi dodaja ali briše uporabnike ter spreminja njihove vloge. Več o urejanju uporabnikov v poglavju 4.8. Entiteta uporabnik je neodvisna od ostalih, zato ni povezana z nobeno izmed ostalih entitet.

Naslednji dve entiteti, ki se na sliki 4.2 nahajata pod uporabnikom, sta učilnica (`Classroom`) in sedež (`Seat`). Medsebojno sta povezani tako, da ima lahko učilnica več sedežev, ki jih hrani v seznamu `seats`, sedež pa pripada eni učilnici in njeno instanco hrani v atributu `classroom`. Vsaka učilnica ima ime (`name`), število vrstic (`rowCount`) in število stolpcev (`columnCount`). Število vrstic in stolpcev določa velikost učilnice, ki predstavlja mrežo sedežev. Vsak sedež ima identifikator vrstice (`row`) in stolpca (`column`) tipa long. Omenjena identifikatorja določata, v kateri vrstici in stolpcu se nahaja sedež v pripa-




Slika 4.2: Entitetni model podatkovne baze

dajoči učilnici. Sedež ima še atributa `ipv4` in `ipv6` znakovnega tipa String, ki predstavljata IP naslov verzije 4 in 6 posameznega sedeža. Atributa sta pomembna pri zajemu goljufij v živo, saj z njima pri vsaki prejeti goljufiji preverimo, če pripadata učilnici, v kateri spremljamo goljufije. Entiteti učilnice in sedeža bi lahko povezali z entiteto goljufije zaradi spremljanja goljufij v živo, vendar smo problem rešili programsko. Zaradi tega je ta povezava nepotrebna, tako da smo pustili učilnico in sedež kot ločen del podatkovne baze. Podrobnejši opis kreiranja sedežnega reda in zaznave goljufij bomo spoznali v kasnejših poglavjih.

Za podrobnejši opis so nam ostale še štiri ključne enitete. To so goljufija (`Cheat`), poskus (`Attempt`), študent (`Student`) in kviz (`Quiz`).

Vsaka goljufija ima IP naslov verzije 4 ali 6 (`ipv4` in `ipv6`) tipa String, iz katerega prepoznamo izvor goljufije. Razlikujemo jih s pomočjo atributa `type`, ki hrani tip goljufije. Primeri tipov goljufij so kopiranje besedila, izguba fokusa brskalnika in podobno. Atribut `additionalData` tipa Text je uporabljen le pri določenih tipih goljufij. Vanj se shranjuje besedilo, ki ga je študent označil. Primer takšne goljufije je kopiranje besedila, pri kateri se v atribut shrani besedilo, ki ga je študent želel kopirati. Atribut `timestamp` predstavlja časovni žig oz. čas, ko je bila izvršena goljufija. Tip časovnega žiga je razred Date, ki hrani čas v milisekundah. Zadnji atribut v entiteti

goljufije je poskus (`attempt`), preko katerega tvorimo povezavo z entiteto poskusa. Prav tako ima en poskus lahko več goljufij, ki so shranjene v seznamu `cheats` tipa List.

Entiteta študent je povezana z entiteto poskusa. Vsak študent ima lahko več poskusov, medtem ko ima en poskus lahko le enega študenta. Vsak študent ima ime in priimek, ki sta združena v atribut (`name`). Ime pridobimo iz sistema Moodle na strani odjemalca. Ker imamo lahko študente z istim imenom in priimkom, jih razlikujemo z atributom `moodleID`. To je identifikacijska številka študenta, ki je uporabljena v sistemu Moodle. Iz entitetnega modela lahko vidimo, da entiteta ne hrani instance razreda posusa, pač pa identifikacijske ključe entitete. Razlog leži v tem, da imamo lahko v Google podatkovni bazi le eno starševsko entiteto. V našem primeru sta starševski entiteti poskusa kviz in študent. Za rešitev problema omejitve smo izbrali za starševsko entiteto kviz, za pravilno relacijo med entitetama študent in poskus pa skrbimo programsko ob vsaki zahtevi v razredih DAO.

Podatke o kvizu hranimo v entiteti `Quiz`. Kviz ima tri ključne znakovne atribute, to so ime (`name`), predmet (`subject`) in povezava na kviz (`url`). V imenu kviza se hrani ime, ki smo ga določili pri kreiranju kviza. Predmet opisuje, pri katerem predmetu se izvaja kviz. Možno je dodati še povezavo na kviz, ki nam kasneje služi kot bližnjica za hitrejši dostop do le-tega.

Entiteta poskus (`Attempt`) predstavlja študenta, ki rešuje kviz v določenem časovnem obdobju. Poskus enolično določata starševski entiteti kviz in študent. Kviz hranimo v atributu `quiz`, ki je instanca razreda `Quiz`, študenta pa v atributu `studentKey`. Študenta ne hranimo v instanci razreda `Student` zaradi omejitve starševskih entitet Google podatkovne baze. Vsak poskus ima dve časovni komponenti: začetek reševanja (`startTime`) in konec reševanja (`endTime`). Oba atributa sta tipa Date in tako shranjena v milisekundah. Sistem Moodle omogoča časovno omejene in časovno neomejene kvize. Iz tega razloga imamo v entiteti poskus atribut `timeLimited`, ki je tipa boolean in nam pove, če je bil poskus reševanja časovno omejen ali ne. Sistem zaznava tudi, iz katerega spletnega brskalnika je študent reševal kviz.

Ime brskalnika se hrani v atributu `browser` tipa String. Ker smo razvili sistem za zaznavanje goljufij, jih moramo tudi hraniti. Vsaka goljufija spada v en poskus, poskus pa hrani goljufije v seznamu `cheats`. Sklop zadnjih štirih opisanih (`startTime`, `endTime`, `browser` in `cheats`) nam omogoča pregledno predstavitev podatkov na časovnici poskusa, opisani v poglavju 4.7.2.

## 4.3 Kreiranje kvizov

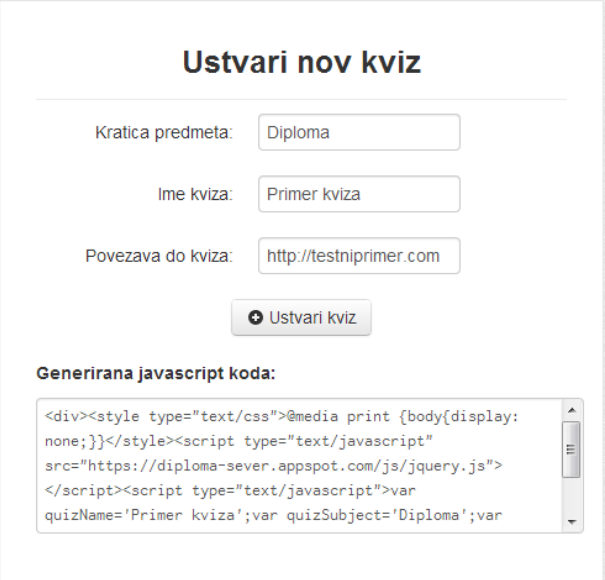
Zaznave goljufij ni možno opisati pred pojasnitvijo tega, kako dosežemo, da iz sistema Moodle pošiljamo goljufije na strežnik sistema. V tem poglavju bomo opisali kreiranje novega kviza. Postopek je razdeljen na dva dela. Najprej se na samem sistemu pridobi kodo v obliki elementa HTML, ki vsebuje programsko kodo Javascript. Nato je potrebno dobljeno kodo uvoziti v sistem Moodle.

### 4.3.1 Generiranje skripte

Postopek se prične na spletni strani sistema. V navigacijski vrstici izberemo opcijo "Ustvari kviz". Odpre se nam spletna stran, ki vsebuje prazno vnosno formo, ki jo prikazuje slika 4.3. V prvo vnosno polje vnesemo predmet oz. kratico predmeta, pri katerem se kviz izvaja. V drugo vnosno polje vnesemo ime kviza, v tretje pa URL naslov, preko katerega lahko dostopamo do kviza. Po vnosu informacij o kvizu v vnosna polja kliknemo na gumb "Ustvari kviz". Kot lahko vidimo na sliki 4.3, se pod izpolnjeno formo prikaže tekstovno polje z generirano Javascript kodo.

Za lažje razumevanje generirane skripte jo bomo opisali po posameznih delih na primeru. Za primer bomo razpolagali z naslednjimi podatki o kvizu:

- Kratica predmeta: `Diploma`
- Ime kviza: `Primer kviza`
- Povezava do kviza: `http://testniprimer.com`




Slika 4.3: Zajem zaslona - generiranje skripte

- Sistem deluje na naslovu: `https://diploma-sever.appspot.com`

```
<div>
  <style type="text/css">@media print {body{display: none;}}</style>
  <script type="text/javascript"
  src="https://diploma-sever.appspot.com/js/jquery.js">
  </script>
  <script type="text/javascript">
    var quizName='Primer kviza';
    var quizSubject='Diploma';
    var quizUrl='http://testiprimer.com';
  </script>
  <script type="text/javascript"
    src="https://diploma-sever.appspot.com/js/cheatDetection.js">
  </script>
</div>
```

Odsek kode 4.1: Sistemsko generirana koda HTML

Z omenjenimi podatki se zgenerira koda prikazana v odseku kode 4.1. Celotna vsebina je ovita s HTML elementom `div`, ki predstavlja delitev strani. Ker element ne vsebuje teksta ali druge vizualne vsebine (Javascript in CSS

ne štejeta kot vizualna vsebina), je njegova privzeta velikost 0. To pomeni, da je za oči opazovalca spletne strani neviden. Ta element je potreben, sicer sistem Moodle blokira pošiljanje goljufij na strežnik.

Prvi element znotraj delitvenega elementa je element `style`, ki vsebuje kodo za definiranje CSS stila strani. Znotraj elementa za stil strani opazimo CSS poizvedbo `@media print`. Tu se pojavi prva omejitev študenta pri goljufanju med reševanjem kviza. Vse, kar se za poizvedbo nahaja v zavitih oklepajih, določa stil vsebine strani za tiskanje. Iz tega sledi, da bo stil natisnjene strani naslednji: `body{display: none;}`. HTML element `body` vsebuje celotno vsebino strani, `display: none` pa določa, naj ta element ne bo viden. Celotna CSS koda torej pomeni, da skrijemo celotno vsebino spletne strani za namen tiskanja oz. predogled tiskanja. To študentom preprečuje izrabo predogleda tiskanja za morebitno goljufanje.

Naslednji element je element `script`, ki uvozi knjižnico jQuery iz samega sistema. Več o uporabljenih knjižnicah je opisano v poglavju 3.4.

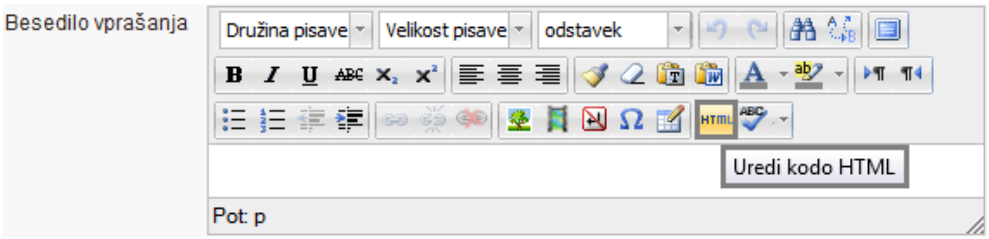
Sledi še en skriptni element, ki vsebuje Javascript spremenljivke. Tu se v tri spremenljivke shranijo vrednosti, ki so bile vnešene v vnosna polja forme. Namen teh spremenljivk je, da ob goljufijah strežniku podamo informacijo o tem, kateri kviz študent rešuje.

Zadnji element generirane programske kode je skriptni element, ki iz sistema uvozi Javascript datoteko `cheatDetection.js`. Uvožena skripta prepozna goljufije in se primerno odzove nanje. Podrobnejši opis zaznave goljufij sledi v poglavju 4.4.

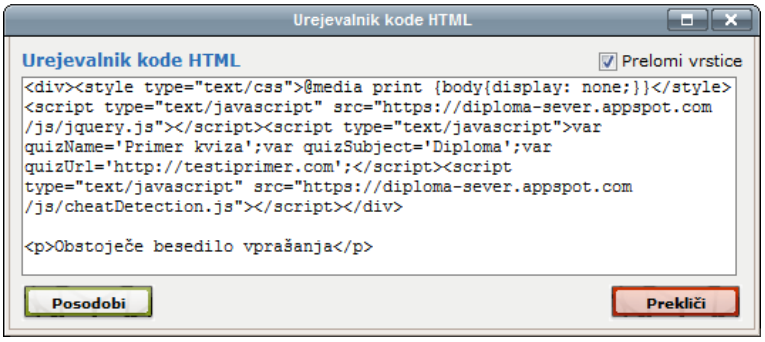
### 4.3.2 Uvoz v sistem Moodle

V prejšnjem poglavju smo spoznali kodo, ki se generira na sistemu. Od nje nimamo nobene koristi, če ostane pozabljena v tekstovnem elementu forme. Potreben je uvoz skripte v sistem Moodle.

Postopek se začne s kreiranjem novega kviza v sistemu Moodle. Ko naredimo nov kviz, lahko začnemo z dodajanjem vprašanj in strani z vprašanji. Pri naslednjem koraku je pomembno to, da na vsako kreirano stran kviza po




Slika 4.4: Urejanje Moodle vprašanja




Slika 4.5: Urejevalnik kode HTML

enkrat uvozimo skripto. Razlog tiči v tem, da ne zaznavamo goljufij preko sistema Moodle, ampak uvažamo svojo Javascript kodo, ki se izvaja na odjemalčevi in ne na strežniški strani. Ker je HTTP protokol, ki ne hrani stanja, bi za preklop med stranmi potrebovali podatek o seji ali piškotku, da bi vedeli podatke o posameznem študentu [33]. To bi lahko dosegli le, če bi izvajali programsko kodo na strežniku. Zaradi te omejitve je potrebno kodo, ki smo jo generirali na sistemu, uvoziti v vsako stran kviza posebej.

Na posamezno stran kviza kodo uvozimo na naslednji način. Izberemo eno izmed vprašanj na strani in ga odpremo v načinu urejanja vprašanja. Nato v polju, ki je označeno z oznako "Besedilo vprašanja", izberemo urejanje v načinu HTML, ki nam omogoča pisanje vprašanja v sintaksi jezika HTML. To naredimo tako, da kliknemo na gumb z oznako "HTML", kot je prikazano na sliki 4.4. Odpre se nam okno za urejanje HTML kode, v njem pa se pokaže besedilo vprašanja. Nad že obstoječe besedilo prilepimo generirano kodo,

ki smo jo pred tem kreirali na sistemu. Končna koda bi morala izgledati podobno kot na sliki 4.5. Na sliki je primer, ki vsebuje besedilo vprašanja "Obstoječe besedilo vprašanja". Pomembno je še enkrat omeniti, da moramo to storiti na vsaki strani kviza.

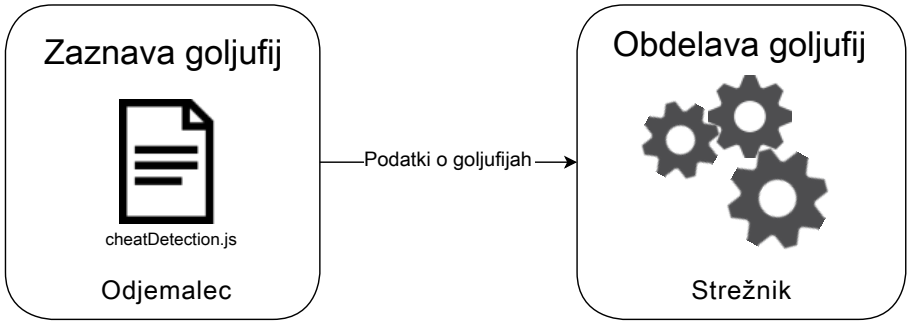
Ob končanem urejanju kviza shranimo spremembe in zaključili smo s kreiranjem kviza. Sedaj se lahko posvetimo zaznavi goljufij, ki je opisana v naslednjem poglavju.

## 4.4 Zaznava goljufij

Najpomembnejša funkcionalnost sistema je proces zaznave goljufij. Na sliki 4.6 vidimo, da je proces sestavljen iz zaznave goljufij na strani odjemalca in obdelave goljufij na strani strežnika. Na strani odjemalca s postopkom kreiranja kviza, opisanim v poglavju 4.3, uvozimo Javascript skripto, ki je zadolžena za zaznavanje goljufij in pošiljanje goljufij strežniku. Strežnik nato goljufijo obdela in shrani v podatkovno bazo. Sledi podrobnejši opis zaznave in obdelave goljufij, saj so stvari kompleksnejše, kot jih prikazuje slika 4.6.

### 4.4.1 Zaznava goljufij na strani odjemalca

Pri zaznavanju vseh vrst goljufij smo zelo omejeni, ker uvažamo svoj sistem v že obstoječi sistem Moodle. Omejitev povzroča Javascript, ki se izvaja na odjemalčevem računalniku. Javascript nam ne omogoča dostopa do sistemskih ukazov, drugače bi lahko izdelovalci spletnih strani imeli dostop in možnost upravljanja vseh računalnikov, katerim nudijo storitve [34]. Tako zaradi omejenosti študentu ne moremo preprečiti izvajanja nekaterih goljufij, lahko pa glede na njegova dejanja dobimo informacijo o poskusu goljufanja. Najenostavnejše si situacijo predstavljamo na preprostem primeru: Pri reševanju kviza ni dovoljeno uporabljati nobene vrste literature. Če torej študent izgubi fokus z oknom, na katerem je prikazan kviz, se to šteje kot goljufija uporabe literature. Tu lahko vidimo glavno težavo - enostavno lahko zaznamo izgubo fokusa, ne moremo pa je preprečiti. V večini primerov povzročamo goljufije




Slika 4.6: Proces zaznave goljufij

s pritiskanjem določenih tipk na tipkovnici, vsa zaznava pa poteka v datoteki `cheatDetection.js`.

Na tem mestu je pomembno omeniti, da ima sistem Moodle dva načina pogleda na kviz. Prvi pogled je reševanje kviza in drugi pregledovanje že rešenega kviza. Skripta zaznava in obdeluje goljufije le v načinu reševanja kviza. To je takrat, ko ima atribut `id` v HTML elementu `body` vrednost enako `page-mod-quiz-attempt`. Če skripta ne zazna omenjenega atributa, se vse "goljufije" ignorirajo.

### Vrste goljufij

Sistem prepoznava vse goljufije povezane z poskusom kopiranja besedila. V to kategorijo spadajo operacije kopiraj, izreži, prilepi in označi vse. Kopiranje shrani označeno besedilo v odložišče operacijskega sistema. Podobno funkcijo ima operacija izreži, ki poleg samega kopiranja označenega besedila v odložišče tega tudi izbriše iz izvornega elementa. Operaciji kopiraj in izreži zaznamo kot goljufijo, saj lahko z njima študent pošlje odgovore na vprašanja, ali vsebino vprašanj, svojim kolegom. Operacija prilepi vsebino iz odložišča operacijskega sistema prilepi v element, v katerega lahko vpisujemo besedilo. Zaznamo jo kot goljufijo, saj lahko študent podaja odgovore na vprašanja iz zunanjih virov in ne z uporabo lastnega znanja, ki ga preverjamo ravno z reševanjem kvizov. Ostala nam je še operacija označi vse, ki omogoča izbiro

celotne vsebine strani. Prav tako jo zaznamo kot goljufijo, saj je za uspešno rešen kviz ne potrebujemo. Če jo kombiniramo z operacijo kopiraj, lahko enostavno kopiramo celotno vsebino spletne strani.

Vse naštete operacije lahko izvajamo z bljižnicami na tipkovnici in s klikom na desni miškin gumb. Bljižnjice na tipkovnici sprožimo s kombinacijo tipke `ctrl` (`command` na računalnikih znamke Apple) in ostalimi tipkami na tipkovnici. Kombinacija tipk za operacijo kopiraj je `ctrl+c`, za izreži `ctrl+x`, za prilepi `ctrl+v` in za označi vse `ctrl+a`. Vse naštete vrste goljufij zaznamo in njihovo izvedbo tudi preprečimo. Za primer vzemimo, da želi študent kopirati besedilo. Goljufijo zaznamo, preprečimo pa tudi kopiranje besedila v odložišče operacijskega sistema, ki ostane v nespremenjenem stanju. Pri akcijah kopiraj, izreži, prilepi in desni miškin klik, skripta shrani tudi označeno besedilo. Kasneje nam to omogoča kakovostnejšo analizo goljufij.

Naslednja možna goljufija je zajem slike zaslona. V tem primeru je drugače kot pri poskusih kopiranja, saj operacije ne moremo zaustaviti. Ob pritisku gumba za zajem zaslona se tako v vsakem primeru zajem shrani v odložišče operacijskega sistema. Uporabna rešitev bi bila brisanje odložišča v določenih časovnih intervalih, vendar to ni mogoče, saj so to sistemske operacije, ki pa jih v Javascriptu ne moremo izvajati. Ker zajem zaslona lahko le zaznamo, lahko goljufijo preprečimo le s strožjimi tolerancami profesorjev do te vrste goljufije.

Dve morda manj očitni goljufiji sta pogled in izvoz izvorne kode strani. Žal sta operaciji dostopni tudi s klikom v menuju brskalnika, ki ga ne moremo preprečiti. Iz tega razloga smo uvedli še zaznavanje izgube fokusa brskalnika. Ta se zazna vsakič, ko uporabnik klikne izven okna brskalnika. Goljufijo za pogled izvorne kode strani lahko prožimo tudi s kombinacijo tipk `ctrl+u`, za shranjevanje izvorne kode pa s kombinacijo `ctrl+s`. Tidve bližnjici lahko zaznamo in onemogočimo njuno izvajanje, zato lahko v vsakem primeru zaznamo guljufijo. Če študent uporabi bližnjico goljufijo zaznamo, ko pa akciji izvrši s pomočjo menija brskalnika, zaznamo izguba fokusa strani.

V sistemu imamo še dve informacijski operaciji, kateri shranjujemo pod

goljufije, vendar ju ne obravnavamo enako. To sta akciji `ready` (nalaganje spletne strani) in `unload` (zapuščanje spletne strani). Prva se sproži takoj, ko se kviz na študentovem računalniku naloži. Podaja nam informacijo o tem, kdaj je študent začel z reševanjem kviza. Druga operacija pa se sproži, če študent odide s spletne strani. Obe torej služita le kot dodatni informaciji o posameznem poskusu reševanja.

### Pošiljanje podatkov na strežnik

Sedaj poznamo vrste goljufij, ki jih s sistemom lahko zaznavamo. Po zaznani goljufiji informacijo pošljemo na strežnik, podrobneje v razred `CheatDetectionServlet.java`, s POST zahtevkom protokola HTTP. Poslanemu zahtevku moramo pripeti še podatke o kvizu, študentu in poskusu. Te pridobimo iz dokumenta DOM.

Podatke o študentu pridobimo ob Javascript dogodku `onready`. Ta se sproži, ko se naložijo vsi elementi spletne strani. Sistem Moodle vsakemu uporabniku sporoča, pod katerim imenom in priimkom je prijavljen v sistem in mu ponuja povezavo na njegov profil in možnost odjave. Vse omenjene informacije se nahajajo v HTML elementu razreda `logininfo`. Primer podatkov študenta Nejca Severja se nahaja v odseku kode 4.2. Iz elementa lahko enostavno razberemo ime in priimek študenta, prav tako njegov Moodle identifikator, ki se nahaja v povezavi `/user/profile.php` kot atribut `id`. V spodnjem primeru je identifikator enak 8627.

```
<div class="logininfo">
  Prijavljeni ste kot
  <a href="https://ucilnica.fri.uni-lj.si/user/profile.php?id=8627">Nejc Sever</a>
  (<a href="https://ucilnica.fri.uni-lj.si/login/logout.php?sesskey=ApO2Yv9o6K">Odjava</a>)
</div>
```

Odsek kode 4.2: Element HTML s prijavnimi podatki študenta

V naslednjem koraku moramo pridobiti lastnosti kviza. Sedaj v upoštev pride kreiranje kviza, opisano v poglavju 4.3. Iz slednjega se spomnimo, da smo lastnosti kviza že zapisali v spremenljivke `quizName`, `quizSubject` in

`quizUrl`.

Potrebujemo le še podatke o časovni omejenosti poskusa in o tem, kdaj se je zgodila goljufija. Čas goljufije preprosto določimo v skripti tako, da ustvarimo Javascript objekt tipa `Date`. Pri določanju omejenosti kviza, pa moramo še bolj podrobno raziskati spreminjanje dokumenta DOM ob reševanju kviza. Opazimo, da se čas hrani v elementu razreda `quiz-timer`, zapisanem v odseku kode 4.3. Na tem mestu se pojavi problem, ker je element vedno viden, vsebina sina `span` pa se spreminja. Če imamo časovno neomejen poskus, potem element razreda `quiz-time-left` ostane prazen, v nasprotnem primeru se vanj zapiše vrednost, ki predstavlja preostali čas reševanja. Problem je v tem, da sistem Moodle preostali čas reševanja generira s pomočjo Javascripta in to za dogodkom, ko se naložijo vsi elementi strani. Tako nismo prepričani, če bo kviz časovno omejen ali ne. Rešitev smo implementirali tako, da skripta po dveh sekundah od nalaganja strani prebere vrednost elementa `span`. Če element vsebuje časovno vrednost, potem imamo informacijo o preostalem času reševanja, v nasprotnem primeru (če je vsebina elementa prazna) privzamemo, da je kviz časovno neomejen.

```
<div id="quiz-timer">
  Preostali cas <span id="quiz-time-left">00:05:45</span>
</div>
```

Odsek kode 4.3: Element HTML s podatki o času reševanja kviza

### 4.4.2 Obdelava podatkov na strani strežnika

Odjemalčeve poslane zahteve sprejemamo v servletu `CheatDetectionServlet.java`. Detekcijo bomo za lažje razumevanje razdelili na tri ločene dele. Prvi del se vrši v omenjenem servletu, drugi del s pomočjo vmesnika Tasks Queue, opisanega v poglavju 3.1.2, v tretjem delu pa se bomo posvetili razredom DAO, ki s pomočjo predpomnenja shranjujejo podatke v bazo.

V zgoraj omenjenem servletu iz zahteve HTTP izluščimo podatke in jih

zapakiramo v objekt tipa JSON. Poleg podatkov, ki jih zajemamo na odjemalčevi strani, potrebujemo še podatke o brskalniku. Te podatke dobimo iz spremenljivke `User-Agent`, ki se nahaja v glavi zahtevka HTTP. Vse podatke nato predamo v FIFO vrsto, kjer se vrši nadaljnja obdelava.

Vmesnik Tasks Queue je FIFO vrsta, ki izvršuje akcije eno za drugo v določenih časovnih intervalih. Za ta Googlov vmesnik smo se odločili zato, ker je ob veliko zahtevah prišlo do konfliktnih transakcij dostopa do podatkovne baze in posledično izgube podatkov. Kot sistem za zaznavo goljufij si ne smemo dovoliti izgube podatkov o goljufijah. Do takšne situacije lahko enostavno pride že ob začetku reševanja. V poglavju 4.4.1 smo spoznali, da se po nalaganju kviza proži dogodek `onready`, ta pa se posreduje strežniku. Predstavljajmo si situacijo, ko vsi študenti ob približno enakem času začnejo reševati kviz. Sproži se veliko zahtevkov ob istem času, kar bi povzročilo izgubo podatkov. Google v svoji dokumentaciji priporoča največ en dostop do enake entitete podatkovne baze v eni sekundi [35], zato je tudi v našem sistemu FIFO vrsta nastavljena na način, da obdeluje eno zahtevo na sekundo. Vrsta kliče servlet `AddCheatServlet.java`, ki poskrbi za dodajanje podatkov v podatkovno bazo. Zaradi počasnejšega pisanja goljufij v podatkovno bazo preko FIFO vrste imamo zakasnitev podatkov pri analizi goljufij. Tako je priporočljivo ob reševanju kviza uporabljati zajem goljufij v živo iz poglavja 4.6, za kasnejšo analizo pa podrobnejšo analizo iz poglavja 4.7. Če imamo veliko študentov in goljufov se lahko zgodi, da bo FIFO vrsta nekoliko v zaostanku s shranjevanjem vseh goljufij. Žal se tej omejitvi ne moremo izogniti drugače kot z uporabo zaznave goljufij v živo, saj brez uporabe vrste pride do konfliktov pri transakcijah.

Na tem mestu smo poskrbeli še za eno optimizacijo sistema. Za dodajanje entitet v podatkovno bazo so zadolženi razredi DAO. Ob shranjevanju kviza, študenta in poskusa pa ne gre zgolj za dodajanje podatkov, ampak je prej potrebno pogledati, če entitete z istimi podatki morda že obstajajo v bazi. Na podlagi tega podatka določimo, če entiteto shranimo ali ne. Ker Google App Engine šteje dnevne kvote (več o dnevnih kvotah v poglavju 4.9.2)

za operacije branja iz podatkovne baze, je dobro le-te omejiti. Zato smo uporabili Googlov predpomnilniški vmesnik Memcache. Ta vmesnik lahko shranjuje javanske razrede v zgoščeni tabeli. Enolični znakovni ključ določa en objekt v zgoščeni tabeli. V našem primeru v predpomnilniku hranimo instance entitet kviza, študenta in poskusa. Tako lahko pred vsakim branjem iz podatkovne baze najprej iz predpomnilnika preberemo, če morda entiteta že obstaja in jo v tem primeru vrnemo. Če entiteta ne obstaja, sledi vpogled v podatkovno bazo. Če entitete ni niti v predpomnilniku niti v podatkovni bazi, dodamo novo. Tako pridobimo na hitrosti in omejimo bralne operacije do podatkovne baze.

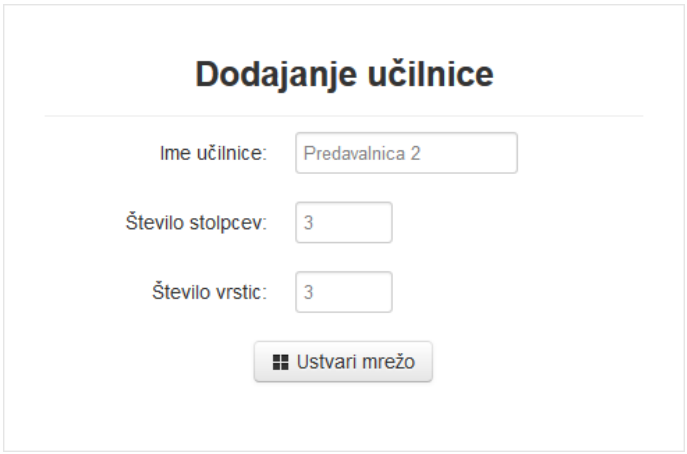
Natančnega podatka o tem, koliko bralnih operacij prihranimo, ne moremo podati, saj se predpomnilnik lahko kadarkoli izprazni [17]. Privzemimo neskončni predpomnilnik, ki se nikoli ne izbriše. Vedno se ob prvem zahtevku ustvari nova instanca razreda entitete, katere obstoj pregledujemo. Ta se najprej shrani v podatkovno bazo, nato še v predpomnilnik. Vsa nadaljnja branja nato potekajo iz predpomnilnika. Največ prihranimo pri entiteti kviza, manj pri entiteti študenta, najmanj pri entiteti poskusa.

## 4.5 Urejanje učilnic

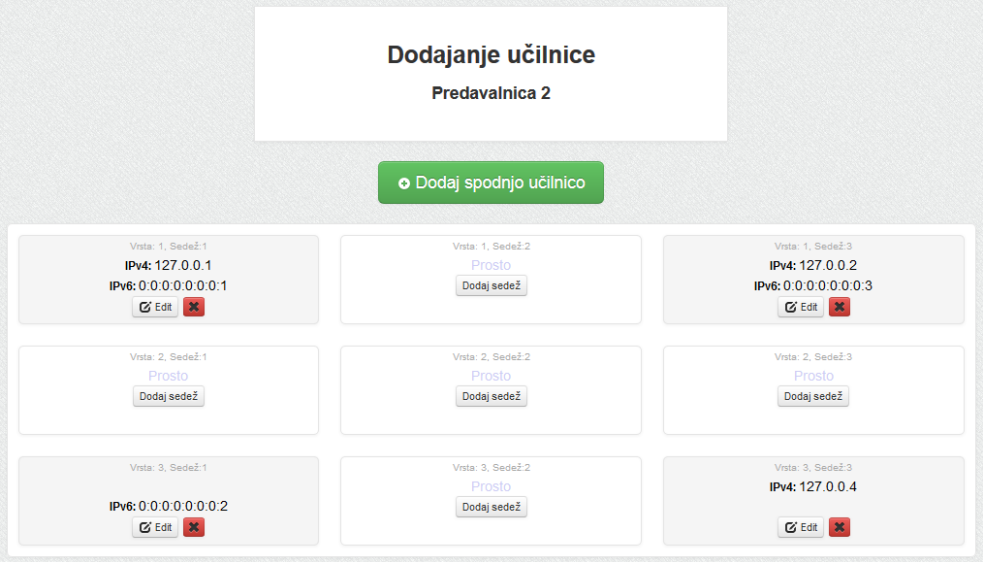
Funkcionalnost dodajanja sedežnega reda učilnic potrebujemo, ker učilnice uporabljamo za zajem goljufij v živo. Zato smo razvili urejevalnik, ki omogoča enostavno dodajanje in urejanje učilnic. Do upravljanja z učilnicami pridemo s klikom na gumb "Učilnice"v navigacijskem meniju strani. Odpre se nam seznam že obstoječih učilnic, kot je prikazan na sliki 4.7. Iz seznama lahko izberemo katerokoli učilnico za izbris ali urejanje.

Dodajanje nove učilnice sprožimo s klikom na gumb "Dodaj novo učilnico" pod seznamom obstoječih učilnic. Odpre se nam vnosna forma, prikazana na sliki 4.8, ki zahteva vnos imena učilnice, števila vrstic in števila stolpcev. Nato kliknemo na gum "Ustvari mrežo", ki sproži generiranje HTML tabele podane velikosti s praznimi sedeži. Za dodajanje sedežev kliknemo na gumb


Slika 4.7: Seznam obstoječih učilnic



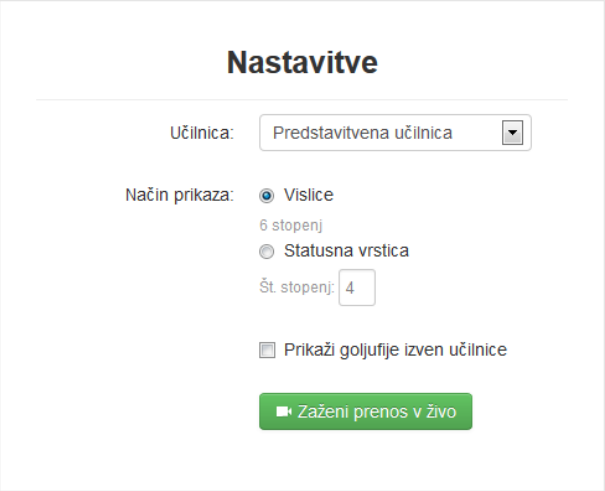

Slika 4.8: Forma za vnos podatkov o učilnici




Slika 4.9: Urejevalnik učilnice

"Dodaj sedež"na mestu, kjer ga želimo dodati. Sedežu lahko določimo naslov IP verzije 4 in verzije 6, ki predstavlja naslov računalnika v učilnici. Urejevalnik je dovolj fleksibilen, da omogoča uporabniku ponazoriti tloris željene učilnice. Če pustimo kakšno vrsto ali stolpec prazen, se ta ob prikazu pokaže z določenim razmakom. Zajem zaslona med samim urejanjem učilnice je prikazan na sliki 4.9. Ko je uporabnik vnesel vse sedeže učilnice, klikne na gumb "Dodaj spodnjo učilnico" in zahteva se pošlje na strežnik. Tam se preko razredov DAO najprej dodajo vsi sedeži, nato še učilnica.

Vsa implementacija funkcionalnosti urejevalnika sloni na knjižnici jQuery. Med urejanjem učilnice imamo definirano globalno spremenljivko, ki je objekt tipa JSON. Ta hrani ime učilnice, število vrstic in stolpcev ter vse aktualne sedeže. Ko uporabnik spreminja postavitev sedežnega reda, se istočasno spreminja tudi vsebina objekta JSON. Ob končanem urejanju se objekt pošlje strežniku, ki prebere vrednosti in učilnico doda oz. posodobi.




Slika 4.10: Nastavitve zajema goljufij v živo

## 4.6 Spremljanje goljufij v živo

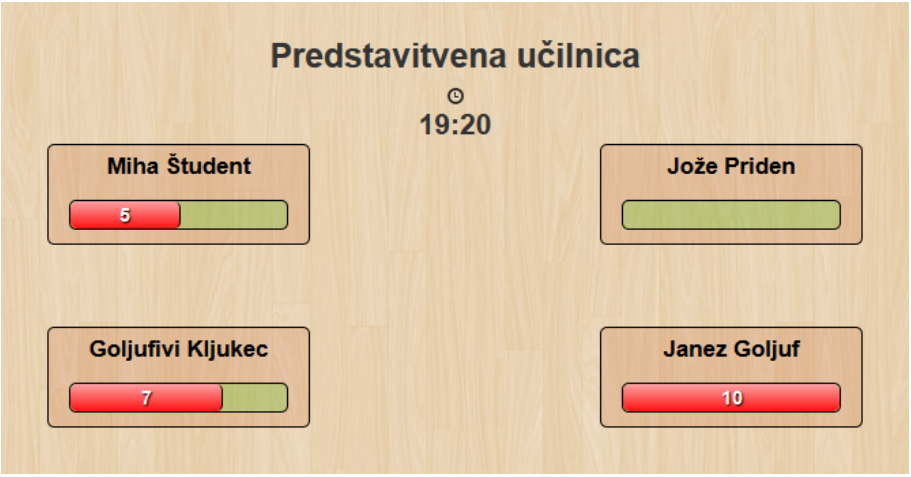
Funkcionalnost zajema goljufij v živo je mišljena predvsem za en način uporabe, to je projeciranje sedežnega reda na projekcijsko platno med reševanjem kviza. Tako imamo pred vsemi prikazan sedežni red učilnice, imena študentov, ki trenutno rešujejo kviz in število goljufij posameznika.

Preden začnemo z zajemom goljufij v živo, moramo kreirati vsaj eno učilnico. To naredimo po postopku, opisanem v poglavju 4.5. Slika 4.10 prikazuje različne opcije za izbiro prikaza sedežnega reda. Goljufije lahko vizualno prikažemo na dva različna načina. Prvi način je z igro vislic, pri katerem se za vsako zaznano goljufijo izvrši ena stopnja vislic. V tem načinu je možno prikazati le 6 goljufij. Druga možnost prikaza je statusna vrstica. V tem načinu lahko število dovoljenih goljufij vnesemo ročno za v to namenjeno vnosno polje. Z vsako goljufijo se zapolni del statusne vrstice, prikazuje pa tudi število izvršenih goljufij posameznega študenta. Prikaz sedežnega reda vislic je prikazan na sliki 4.11, prikaz s statusno vrstico pa na sliki 4.12. Poleg vizualne predstavitve števila goljufij posameznega študenta imamo tudi možnost prikaza goljufij izven učilnic. To pomeni, da v primeru, ko zaznana


Slika 4.11: Prikaz goljufij - vislice

goljufija ne spada pod noben sedež (naslov IP ne pripada nobenemu sedežu), programsko dodamo nov sedež pod vizualizacijo učilnice.

Za implementacijo zajema v živo smo uporabili Google App Engine vmesnik Channel API z manjšo nadgradnjo. V osnovi je vmesnik namenjen pošiljanju sporočil po kanalu, ki se vzpostavi med dvema odjemalcema. Odjemalci lahko poslušajo ali pošiljajo sporočila preko strežnika s ključi kanala, ki služijo kot enolični identifikator kanala. Odjemalec pošlje sporočilo tako, da poda ključ kanala, po katerem želi pošiljati sporočilo, in vsebino sporočila. Ko strežnik prejme sporočilo, pošlje sporočilo vsem klientom, ki poslušajo na kanalu, določenim s prejetim ključem [17]. To lastnost lahko izkoristimo za zaznavo goljufij. Sprememba je le v tem, da sporočila ne tvori odjemalec ampak kar strežnik sam. Odjemalec ob zagonu spremljanja goljufij v živo strežniku sporoči, da želi od njega prejemati vse goljufije, ki jih bo v prihodnosti zaznal. Strežnik za vsakega novega odjemalca ustvari ključ kanala ter mu ga posreduje. Ključ shrani tudi v aplikacijsko spremenljivko, v kateri




Slika 4.12: Prikaz goljufij - statusna vrstica

hrani seznam ključev vseh aktivnih kanalov. Odjemalec prejme ključ, ki ga je generiral strežnik, in začne s poslušanjem goljufij na kanalu, ki ga določa ključ. Strežnik se ob vsaki zaznani goljufiji sprehodi čez zeznam shranjenih ključev v aplikacijski spremenljivki in na vsak kanal pošlje podatke o goljufiji. Ko odjemalec zapusti stran zajema goljufij, se pošlje strežniku zahteva, ta pa izbriše ključ iz seznama in kanal se izbriše.

Ko imamo vzpostavljeno povezavo s strežnikom, moramo prejete goljufije uvrstiti na pravo mesto v učilnici. Uvrščanje se izvaja na odjemalcu s pomočjo Javascript skripte. Ob nalaganju strani se izvede povezava s kanalom, kot je opisano v prejšnjem odstavku. Poleg ključa kanala odjemalec prejme tudi podatke o učilnici in njenih sedežih. Iz teh podatkov ustvarimo vizualno podobo učilnice ter dve zgoščeni tabeli. Ključi elementov obeh tabel so IP naslovi vseh sedežev v učilnici. V prvi tabeli so IP naslovi verzije 4, v drugi verzije 6. Vrednost elementov v tabelah je ime študenta, stolpec in vrstica sedeža ter števec goljufij od začetka zajema prenosa. Vsaka prejeta goljufija im svoj IP naslov izvora. Če naslov pripada kakšnemu izmed ključev v zgoščenih tabelah, potem vemo, kateremu mestu v ucilnici goljufija pripada. Na tem mestu povečamo števec goljufij in osvežimo prikaz sedeža.

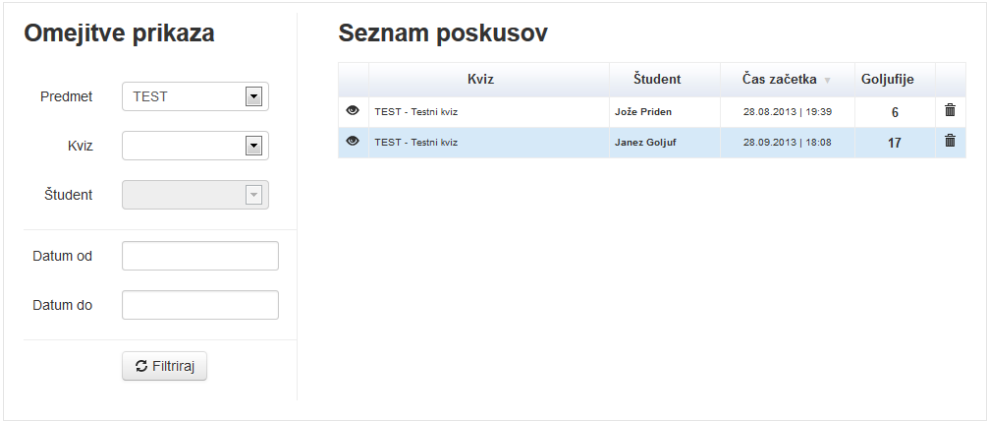
## 4.7 Analiza goljufij

Za podrobnejšo analizo goljufov je dobro imeti organiziran način prikaza zaznanih goljufij in poskusov reševanj kvizov. Sistem nam omogoča pregled goljufij v dveh korakih. Prvi korak je filtriranje in izbira poskusov glede na podane parametre. Iz filtriranih poskusov lahko nato izberemo posamezen poskus za podrobnejšo obravnavo.

### 4.7.1 Filtriranje poskusov

V tem poglavju bomo opisali prvi korak pregledovanja goljufij. Na sliki 4.13 je zajem zaslona, ki prikazuje element glavne strani za analizo goljufij. Na levi strani pod naslovom `Omejitve prikaza` se nahajajo polja za vnos filtrov, na desni strani pa je tabela vseh poskusov, ki so rezultat vnešenih filtrov na levi.

Filtri poskusov so razdeljeni na dva dela. Prvi del (na sliki 4.13 levo zgoraj) je sestavljen iz treh izbirnih polj. Prvo je za izbiro predmeta, drugo za ime kviza in tretje za ime študenta. Zadnji dve sta odvisni od izbire predhodnih polj. Ob prihodu na spletno stran je omogočeno le izbirno polje za izbiro predmeta. Tu je seznam vseh predmetov, pri katerih je bil kadarkoli reševan kviz. Ob izbiri enega izmed predmetov se za izbrani predmet napolni izbirno polje za ime kviza. Tu se izpišejo vsa imena kvizov, ki pripadajo izbranemu predmetu. Ob izbiri imena kviza pa se napolni še izbirno polje za ime študenta. Tu so vsi študenti, ki pripadajo zgornjim dvem pogojem. Vse zahteve po imenih se izvajajo asinhrono s pomočjo knjižnice jQuery in zahtevkov AJAX, preko katerih prejmemo navaden seznam, ki vsebuje vrednosti za napolnitev izbirnih menujev. Izbira omenjenih filtrov je poljubna, zato jih lahko pustimo tudi prazne. V tem primeru so v tabeli na desni strani okna prikazani poskusi vseh predmetov, kvizov in študentov. Drugi način filtriranja poskusov je preko izbire časovnega intervala. Za vpis datumov imamo na voljo dve polji (na sliki 4.13 levo spodaj), "datum od" in "datum do". Rezultat tega filtra so vsi poskusi, ki se začnejo v izbranem časovnem in-




Slika 4.13: Seznam poskusov reševanja

tervalu. Za lažji vnos imamo ob kliku na voljo izbirno okno, v katerem s pomočjo koledarja in drsnikov nastavimo datum in čas [29]. Vnos datumov je prav tako kot izbirna polja poljuben. Za potrditev filtrov kliknemo na gumb "Filtriraj" in rezultati filtrov so vidni v tabeli na desni strani.

Za lepši prikaz poskusov smo uporabili orodje Google Charts [28], podrobneje Table Charts [36]. To je vmesnik, ki nam omogoča vnos podatkov v tabelo preko objekta JSON, omogoča pa tudi urejanje vrstnega reda vrstic v ustvarjeni tabeli. V tabeli poskusov imamo prikazane podatke o imenu kviza (s kratico premeta kot predpono), imenu študenta, času začetka kviza in številu goljufij. Vse podatke dobimo ob nalaganju strani s strežnika in jih na strani JSP vnesemo v objekt JSON, ki ga vmesnik prejme kot vhodni parameter. Na levi strani je prikazana ikona očesa, ki ob kliku omogoča pogled časovnice poskusa, opisane v poglavju 4.7.2. Na desni strani se nahaja ikona koša, ki ob kliku izbriše izbran poskus iz podatkovne baze. Poskusi so privzeto razvrščeni glede na čas reševanja, od najmlajšega do najstarejšega. S klikom na glavo stolpca lahko razvrstimo vrstice glede na poljuben stolpec, naraščujoče ali padajoče.

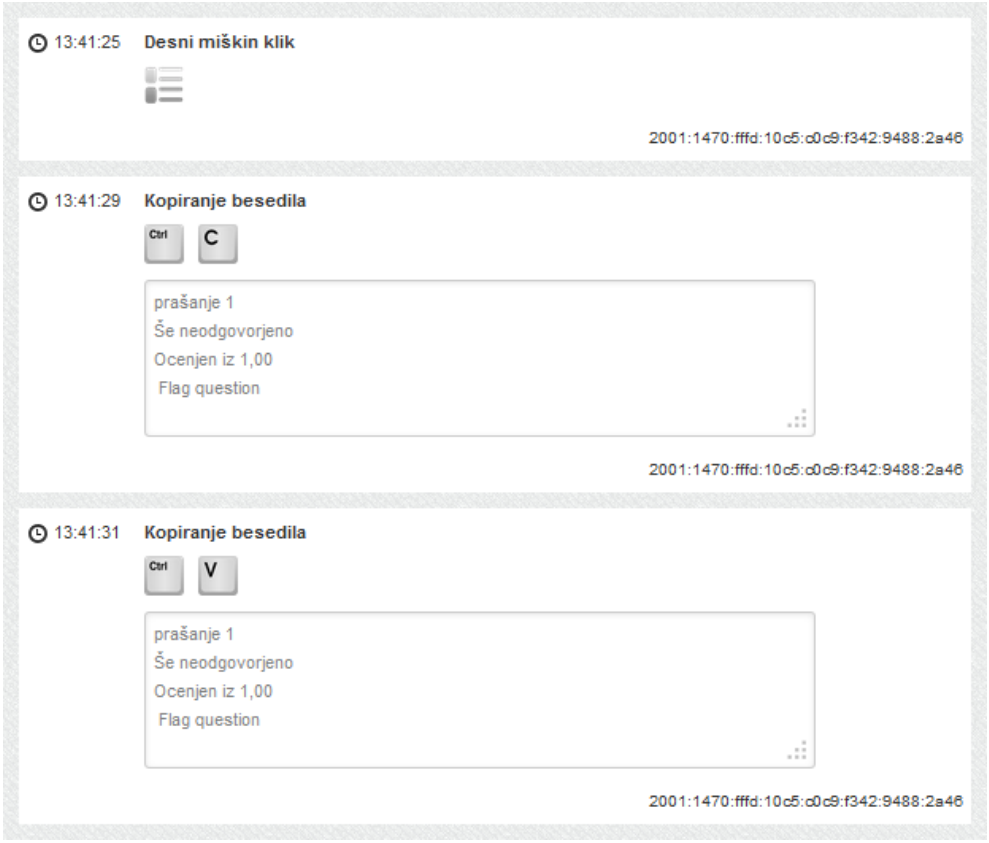

Slika 4.14: Podatki o poskusu

### 4.7.2 Časovnica poskusa

S klikom na ikono očesa v tabeli poskusov si lahko ogledamo časovnico poskusa. Ta nam prikazuje vse zaznane goljufije in njihove lastnosti. Časovnica je razdeljena na del s splošnimi podatki poskusa in del s seznamom goljufij. V nasprotju z zaznavanjem goljufij v živo, kjer imamo vpogled le v število goljufij posameznika, nam časovnica omogoča pregled vseh podrobnosti goljufij.

Okvir s podatki poskusa je prikazan na sliki 4.14. Na vrhu se nahaja ime kviza, združeno s predmetom. Če je uporabnik sistema ob kreiranju kviza podal povezavo do kviza, lahko s klikom naslova pridemo na cilj povezave. Pod naslovom se nahaja študentovo ime, pod imenom pa čas začetka reševanja oz. zaznave prve goljufije ter čas končanja poskusa. Čas končanja lahko ostane prazen, če je kviz brez časovne omejitve. Prikazana je tudi ikona brskalnika, preko katerega je študent reševal kviz, ter število vseh goljufij.

Pod splošnimi podatki poskusa so v seznamu vse goljufije, razvrščene po času od najstarejše do najmlajše. Na sliki 4.15 je primer treh goljufij. Posamezen okvir goljufije sestoji iz naslednjih podatkov: na levi strani je prikazan čas goljufije, desno od časa je krepko napisan opis goljufije, pod opisom pa




Slika 4.15: Časovnica poskusa

ikona bližnjice goljufije (primer kopiranja: `ctrl+c`). V desnem spodnjem kotu je prikazan naslov IP, iz katerega je prišla goljufija. V primeru s slike je naslov IP verzije 6. Opazimo, da je pri nekaterih okvirjih prikazano tudi tekstovno polje. V to polje je vnešena vrednost atributa `additionalData` iz podatkovne baze, ki smo ga omenili v poglavju 4.2.1 in je značilen le za določene tipe goljufij. Če je vrednost atributa prazna, potem tekstovnega polja ne prikažemo. V časovnici se prikazujejo tudi informativne goljufije, kot sta nalaganje strani in odhod s strani, za boljšo predstavo o tem, kako je potekalo reševanje kviza.

## 4.8 Uporabniki sistema

Ciljna množica uporabnikov sistema za zaznavo goljufij na spletnih kvizih so predvsem predavatelji na fakultetah in učitelji na šolah. Ker ne želimo, da bi imeli ostali uporabniki dostop do funkcionalnosti, je sistem zaščiten. Za vstop je tako potrebna prijava, ki poteka s pomočjo vmesnika Google Users API. To je storitev za prijavo uporabnikov preko Google računov, ki nam močno olajša implementacijo overjanja uporabnikov. Vmesnik omogoča dostop do strani vsem uporabnikom, ki imajo Googlov račun, česar v našem primeru ne želimo. Za dodatno varnost smo poskrbeli tako, da shranjujemo elektronske naslove uporabnikov, ki jim je dovoljena prijava v sistem. Za upravljanje s podatki o dovoljenih uporabnikih zato potrebujemo administrativen dostop.

Za dodajanje in brisanje uporabnikov ni implementiranega grafičnega vmesnika, so pa na voljo kratka navodila uporabe. Do seznama uporabnikov lahko dostopamo preko naslova `<strežniška_domena>/users`. Za primer vzpostavitve sistema, to je ko v bazi še ni podatkov o uporabnikih, je ta naslov dostopen vsem uporabnikom. Namen tega je, da ob vzpostavitvi sistema določimo administratorja. Ko je v sistemu prisoten vsaj en uporabnik, lahko do naslova dostopajo samo uporabniki z elektronskimi naslovi, ki jih je dodal administrator.

Uporabnike prav tako dodajamo preko nalova `/users`, le da mu podamo parametre o elektronskem naslovu in uporabniški vlogi. Vloga uporabnika je lahko administrator (`admin`) ali navaden uporabnik (`user`). Le administratorjem je dovoljeno urejati uporabnike, zato moramo biti pazljivi, da prvemu dodanemu uporabniku določimo vlogo `admin`. Parametre za dodajanje uporabnika dodamo kar za naslov v obliki `/users?action=add&email=[uporabnikov-email]&role=[vloga]`. Parameter `action` ima vrednost `add`, kar označuje akcijo dodajanja uporabnika, parameter `email` določa elektronski naslov, vlogo pa parameter `role`. Ob zahtevku HTTP tipa GET navedenega naslova se podani podatki o uporabniku shranijo v podatkovno bazo. Brisanje uporabnika poteka na enak način, le brez podajanja vloge in z atributom `action=remove`. Naslov za brisanje uporabnika je v obliki

`/users?action=remove&email=[uporabnikov-email]`.

Sistem za funkcionalnosti izrablja strani JSP in Servlete, zato overjanje uporabnikov poteka na vseh straneh JSP in na vseh Servletih. Če uporabnik ni prijavljen, je preusmerjen na prijavno spletno stran.

## 4.9 Podpora in omejitve

Razširjenost uporabe sistema se zelo poveča, če ga implementiramo tako, da podpira napopularnejše uporabljene spletne brskalnike in operacijske sisteme. Na podporo je potrebno računati že v fazi načrtovanja sistema, saj tako vnaprej vemo, katere tehnologije bomo uporabili v fazi implementacije sistema. V poglavju bomo opisali, katere brskalnike in operacijske sisteme podpira sistem in kako smo podporo zagotovili. Opisali bomo tudi, kaj so dnevne kvote strežnikov Google App Engine in v kolikšni meri jih sistem zasede.

### 4.9.1 Podpora brskalnikov in OS

Sistem temelji na spletnih tehnologijah, zato je glavni povdarek v podpori brskalnikov in manj na podpori operacijskih sistemov. Pri podpori operacijskih sistemov moramo biti pozorni le pri različnih vrstah tipk računalnikov pri zaznavanju goljufij. Sistem mora razlikovati med tipkama `ctrl` in `cmd` na računalnikih znamke Apple oz. na operacijskem sistemu iOS.

Za podporo vseh brskalnikov je potrebno vložiti več truda. To za najpogosteje uporabljene brskalnike omogoča knjižnica jQuery. To je knjižnica funkcij, napisana v jeziku Javascript, ki preverjeno delujejo na pogosto uporabljenih brskalnikih [25]. Za enoten izgled sistema smo uporabili knjižnico bootstrap, opisano v poglavju 3.4, ki nam omogoča pregledno postavitev elementov in lep izgled gradnikov spletnih strani. Knjižnica podpira tudi spreminjajočo postavitev elementov za mobilne naprave [27], ki je nismo uporabili, saj je poudarek diplomske naloge predvsem v implementaciji zaznave in analize goljufij. Potrebno je pogledati še iz zornega kota sistema Moodle in

Slika 4.16: Podprti brskalniki

Slika 4.17: Podprti operacijski sistemi

reševanja kvizov. Knjižnica jQuery nam omogoča pošiljanje zahtevkov AJAX in procesiranje dokumenta DOM spletnega kviza brez dodatnih problemov na večini brskalnikov. Pojavila se je le ena težava pri uporabi brskalnika Google Chrome. Ker sistem Moodle uporablja varnejšo obliko protokola HTTP, to je HTTPS, brskalnik Google Chrome zahteva vse klice iz odjemalca s tem protokolom. Zato je potrebno ob vzpostavitvi projekta na Google App Engine omogočiti povezave s protokolom HTTPS in ob kreiranju kviza podati HTTPS naslov sistema. S to nastavitvijo smo odpravili še zadnjo težavo podpore sistema na večini brskalnikov. Logotipi brskalnikov, ki jih sistem preverjeno podpira, so na sliki 4.16 od leve proti desni: Google Chrome, Opera, Internet Explorer, Mozilla Firefox in Safari. Logotipi podprtih oprecijski sistemov pa so na sliki 4.17 od leve proti desni: Microsoft Windows, Linux in iOS.

### 4.9.2 Google dnevne kvote

Sistem smo implementirali tako, da ga poganja Google App Engine. Za zagotavljanje brezplačnih storitev je potrebno upoštevati dnevne kvote. Re-

zultat prekoračitve dnevnih kvot je plačilo za koriščenje storitev [12]. Ker je vrst dnevnih kvot veliko, bomo opisali le tiste, ki so pomembne za delovanje našega sistema.

### Čas streženja instanc sistema

Dnevna kvota čas streženja instanc sistema (angl. Frontend Instance Hours) predstavlja streženje vsebine, kot so skripte, spletne strani in vse potrebne spletne vsebine. Omejeni smo na število ur streženja storitev na dan. Da nam za prekoračitev kvote ni potrebno plačati, mora biti ta čas manjši od 28 ur na dan. Nova instanca se generira, če je v nekem trenutku preveč zahtevkov na strežnik, da bi jih obdelovala le ena. Več instanc lahko teče istočasno, čas streženja storitev pa se medsebojno sešteva. Google samo za vzpostavitev in opuščanje instance računa 15 minut, temu pa se prišteje čas streženja. Če torej storitev strežemo 10 minut, se nam v skupni seštevek šteje 10 minut streženja in 15 minut za vspostavitev, skupno 25 minut [37].

V našem primeru ta dnevna kvota ne predstavlja težav, saj je sistem obremenjen le med reševanji kvizov in analizo goljufij. Razmerje med zasedenostjo sistema in številom ur v dnevu je majhno, zato je manjša verjetnost, da bi prekoračili dnevno kvoto.

### Količina podatkov v podatkovni bazi

Brezplačni dnevni limit količine shranjenih podatkov v podatkovni bazi je 1 GB [38]. Ker lahko velikosti entitet malenkost variirajo, ne moremo podati točne številke, lahko pa ocenimo, koliko prostora porabimo z našimi entitetami. Za test smo imeli v podatkovni bazi skupno shranjenih 351 entitet. Od tega je bilo 5 entitet kvizov, 2 entiteti študentov, 5 uporabnikov, 7 poskusov, 3 učilnice, 20 sedežev, ostalo pa so bile entitete goljufij. Vsi našteti podatki so zavzemali 829 KB prostora. Če bi želeli doseči dnevno kvoto, bi morali hraniti približno 75 tisočkrat več podatkov. Težko si predstavljamo, da bomo zaznali toliko goljufij ali ustvarili toliko novih učilnic v enem dnevu, zato tudi tej dnevni kvoti zadoščamo z veliko rezerve.

### Število pisalnih operacij

Dnevna kvota predstavlja število pisanj v podatkovno bazo. V našem primeru je najpogostejši zapis v podatkovno bazo ravno entiteta goljufije. Google za brezplačne storitve zahteva pod 50000 dnevnih pisalnih operacij [38]. Tudi tukaj smo daleč od dosega dnevne kvote. Pol dneva ima 43200 sekund, kar pomeni, da bi morali pol dneva zavzeto goljufati, če bi se želeli približati dnevni kvoti.

### Število bralnih operacij

Pomembna kvota, ki tudi za naš sistem povzroča težave, so bralne operacije nad podatki iz podatkovne baze. Bralnih operacij je na voljo toliko kot pisalnih, torej 50000 operacij na dan [38]. Največ jih porabimo ob analizi goljufij, natančneje pri ogledu časovnice poskusa in filtriranju poskusov. Z uporabo entitet JPA se iz podatkovne baze pri branju poskusov naložijo tudi vse goljufije poskusov, kar prispeva k večjemu številu bralnih operacij. To je edina kvota, ki bi se v primeru zelo velikega števila goljufij lahko približala brezplačni dnevni rabi. Bralne operacije uporabljamo tudi pri zajemu goljufij, opisanem v poglavju 4.4, kjer smo opisali uporabo predpomnilnika za zmanjševanje dostopov do podatkovne baze ravno iz razloga omejitve dnevnih kvot.

### Velikost naloženih datotek

Kvota je namenjena omejitvi velikosti datotek, ki jih naložimo na strežnik. Ta kvota je za razliko od ostalih stalna in ne dnevna. Za brezplačno delovanje aplikacije je dovoljeno naložiti do 1 GB podatkov [38]. V grobem naš sistem znaša 70 MB, kar je daleč pod mejo, zato s to kvoto nismo omejeni.

### Klici vrste za izvajanje operacij

Task Queue API Calls pomeni število klicev, ki jih izvedemo na FIFO vrsto. V poglavju 4.4 smo opisali, zakaj smo uporabili vmesni Tasks Queue API.

Google nam brezplačno zagotavlja 100000 klicev vrste na dan, kar je več kot dovolj za uporabo našega sistema. Ker uporabljamo vrsto le za zaznavo goljufij, lahko ponovno podamo primerjavo o številu potrebnih goljufij, da dosežemo omejitev. Dan ima 86400 sekund, kar pomeni, da bi morali povzročiti goljufijo več kot enkrat na sekundo v enem dnevu. Vemo, da je to skoraj nemogoče, zato ta kvota ne predstavlja težav.

### Število ustvarjenih kanalov

Zadnja pomembna kvota so kanali, ustvarjeni s pomočjo vmesnika Channels API. Vmesnik uporabljamo pri zajemu goljufij v živo, preko katerega se odjemalec poveže na strežnik, ta pa mu posreduje zaznane goljufije. Vsakič, ko zaženemo spremljanje goljufij v živo, ustvarimo nov kanal. Google za brezplačno uporabo storitev ponuja 100 ustvarjenih kanalov na dan [38]. V našem primeru to pomeni, da lahko 100krat na dan zaženemo spremljanje goljufij v živo. Morda se kdaj lahko približamo tej meji, vendar bi tudi 100 kanalov moralo več kot zadostovati.

# Poglavje 5

# Sklepne ugotovitve

V diplomski nalogi smo opisali uporabljene tehnologije in implementacijo sistema za podporo reševanja spletnih kvizov. Izpolnili smo primarni cilj diplomske naloge, ki je razvoj delujočega prototipa takšnega sistema. Ta omogoča zaznavamo in preprečevanje goljufij na strani odjemalca, spremljanje goljufij v eni izmed kreiranih učilnic in pregledovanje časovnice poskusov posameznikov. Sistem je kompatibilen z in testiran na najpogosteje uporabljenih operacijskih sistemih in brskalnikih, zato je izkušnja uporabe neodvisna od uporabljene tehnologije. Izvorna koda projekta je dostopna na naslovu `https://github.com/nejcsever/moodleAntiCheat`. Na koncu pisnega dela diplomskega dela bomo opisali še težave, ki so se pojavile pri razvoju in ideje za nadaljnji razvoj projekta.

## 5.1 Težave pri razvoju

Pri implementaciji sistema se nismo mogli izogniti določenim težavam, ki pa smo jih uspešno odpravili. Pri večini težav je bil razlog omejenost tehnologije, s katero smo razvijali sistem.

Prva težava pri zajemu goljufij je bil način nalaganja skripte na odjemalčev računalnik brez potrebe po kakršni koli programski namestitvi. Rešitev smo našli v HTML urejevalniku vprašanj v sistemu Moodle, preko

katerega lahko vnesemo različne HTML elemente. To je omogočalo uvoz datoteke Javascript, s katero zaznamo goljufije in podatke pošljemo na strežnik.

Največ težav smo imeli pri implementaciji strežniškega dela, ki teče na sistemu Google App Engine. Prva je bila v omejenosti podpore vmesnika JPA. Ker Google App Engine ne hrani podatkov v relacijski podatkovni bazi ampak v strukturi BigTable [39], so nekatere funkcionalnosti vmesnika JPA omejene. Težavo je povzročilo dejstvo, da ima lahko ena entiteta le eno starševsko entiteto. Tudi poizvedovalni jezik JPQL na Google App Engine ima določene omejitve. Primer je omejeno število primerjalnih operatorjev v enem JPQL stavku. Za omenjena problema smo našli programski rešitvi, s katerima smo obšli omejitve.

Eden izmed ciljev prototipa diplomske naloge je, da deluje brezplačno. Za zadovoljevanje tega cilja smo morali optimizirati strežniško delovanje, da smo zadovoljili dnevne kvote Google App Engine. Kot smo opisali v poglavju 4.9.2, smo imeli največ težav pri bralnih dostopih do podatkovne baze. To smo rešili z uporabo vmesnika za predpomnenje. To pa ni edina težava, ki smo jo rešili z Googlovimi vmesniki. Kot je bilo v diplomski nalogi že omenjeno, je pri več kot eni prejeti goljufiji na sekundo prihajalo do konfliktnih transakcij in izgube podatkov. Težavo smo uspešno rešili z uporabo še enega vmesnika, to je vmesnik Tasks Queue API.

## 5.2 Možne izboljšave

Pri vsaki programski rešitvi je vedno na voljo prostor za izboljšave, saj redkokateri kompleksni sistemi delujejo brez hroščev. Ker takšen sistem za zaznavo goljufij še ne obstaja, ga je bilo potrebno zgraditi od temeljev do končne rešitve. Najprej je bilo potrebno rešitev načrtovati in vzpostaviti osnovne funkcionalnosti, nato pa še dodatne, kot so analiza goljufij in zajem goljufij v živo. Izboljšave so možne tako na ravni optimiziranja programske kode in njenega delovanja kot na ravni dodatnih funkcionalnosti.

Pravilnost delovanja sistema smo testirali, vendar se zagotovo najde še

kakšna izboljšava programske kode, ki nam je ostala skrita. Izdelek je potrebno začeti uporabljati v šolah in na fakultetah in od njih pridobiti povratne informacije o nepravilnostih, možnih popravkih in hitrosti delovanja ter ideje za nove funkcionalnosti. Za čim optimalnejše delovanje programske kode sistema smo poskrbeli pri načrtovanju in implementiranju, zato v tem trenutku morda ni vidnih veliko izboljšav. Po uporabi produkta v praksi bi se zagotovo našlo nekaj boljših rešitev, ki bi jih lahko uvedli v sistem. V nadaljnjem razvoju je odprta možnost za zmanjšanje števila bralnih operacij iz podatkovne baze. To število vpliva na dnevno kvoto, opisano v poglavju 4.9.2. Za zaznavo goljufij v živo smo jo izboljšali s predpomnenjem, v prihodnosti bi lahko poskrbeli še za optimizacijo problema pri analizi goljufij. Izvorna koda projekta je javno dostopna v sistemu GitHub in tako se lahko nadaljnjemu razvoju pridružijo programerji, ki bi bili zainteresirani za projekt.

Prostora za dodajanje funkcionalnosti je veliko. Cilj diplomske naloge je bil zgraditi prototip zaznave in analiza goljufij ter spremljanja goljufij v živo. V nadaljnjem razvoju bi lahko dodali nekaj tehnik za podrobnejšo analizo goljufij. Sistem podpira analizo poskusov posameznikov, ki so reševali kviz. Dodatno analizo bi lahko vršili na samih kvizih oz. časovnih intervalih določenega kviza. Tako bi lahko določili časovne intervale reševanja kviza, pri katerih je prišlo do največjega števila goljufij. Še boljša analiza bi bila mogoča s kombiniranjem sedežnega reda in zaznanih goljufij, kar bi nam omogočalo zaznavanje medsebojnega sodelovanja med večimi študenti v isti učilnici. Medsebojne goljufije bi lahko ponazorili s prikazom zaznanih goljufij za posamezni sedež v učilnici. Nato bi preverjali pogostost goljufij sosednjih sedežev in kasneje še posamezne poskuse, ki bi se nam zdeli sumljivi. Sistemu bi lahko dodali zaznavanje povezav med akcijami kopiraj in prilepi. Zaznali bi študenta, ki je prilepil enako vsebino besedila v odgovor na vprašanje, kot ga je nek drug študent kopiral iz svojega kviza.

Rešitve pa niso le programske narave. Za ostrejši nadzor nad goljufijami bi lahko uvedli sistem kamer. Študent bi moral imeti ob reševanju nastavljeno kamero, ki bi snemala njega in njegov ekran. Tako bi lahko preverjali

reševanje kviza tudi na daljavo. Sistem bi zaznaval morebitne goljufije, ki jih uporabnik izvaja s pomočjo miške in tipkovnice, kamera pa sam zaslon ter morebitne goljufije, ki ne izvirajo iz računalnika (dodatna literatura, fotografiranje zaslona). Z omenjeno rešitvijo bi lahko nadgradili tudi prikaz sedežnega reda. Na vizualnem prikazu učilnice bi lahko namesto vislic in statusne vrstice prikazovali kar video, ki bi ga snemala kamera, usmerjena v študenta. Ta rešitev bi zahtevala dodatne finančne vložke, bi pa zagotovo pripomogla k manjšemu številu goljufij.

# Literatura


[1] (2013, september) Coursera. Dostopno na: https://www.coursera.org/

[2] (2013, september) Udacity. Dostopno na: https://www.udacity.com

[3] A. Aiken. (2011, junij) Moss - A System for Detecting Software Plagiarism. Dostopno na: http://theory.stanford.edu/~aiken/moss/

[4] Google. (2013, avgust) Google App Engine. Dostopno na: https://developers.google.com/appengine/

[5] N. A. Buzzetto-More, *Principles of Effective Online Teaching*. Informing Science, januar 2007.

[6] Z. Sun. (2013, september) Moss Moodle plugin. Dostopno na: https://moodle.org/plugins/view.php?plugin=plagiarism_moss

[7] iParadigms, LCC. (2013, september) iThenticate Professional Plagiarism Prevention. Dostopno na: http://www.ithenticate.com/

[8] K. Mohile. (2009, julij) What is System Lockdown. Dostopno na: http://www.symantec.com/connect/articles/what-system-lockdown-what-stages-do-i-implement-system-lockdown-symantec-endpoint-protectio

[9] brian. (2008, oktober) Critical Analysis of Respondus LockDown Web Browser. Dostopno na: http://opensource.cse.ohio-state.edu/lockdown

[10] Respondus, Inc. (2013, september) Respondus LockDown Browser. Dostopno na: http://www.respondus.com/products/lockdown-browser/

[11] WebAssign. (2013, september) WebAssign LockDown Browser. Dostopno na: http://www.webassign.net/user_support/student/lockdown_browser.html

[12] Google. (2013, avgust) Paid Apps: Budgeting, Billing, and Buying Resources. Dostopno na: https://developers.google.com/appengine/docs/billing

[13] U. M. in Borut Fabjan, *Java 2, temelji programiranja*. Pasadena, avgust 2004.

[14] Google. (2013, avgust) Google App Engine - Java Runtime Environment. Dostopno na: https://developers.google.com/appengine/docs/java/

[15] A. Goncalves, *Beginning Java EE 7*, 1. izdaja. Apress, junij 2013.

[16] H. Bergsten, *JavaServer Pages*, 3. izdaja. O'Reilly Media, oktober 2009.

[17] Google. (2013, september) Memcache Java API Overview. Dostopno na: https://developers.google.com/appengine/docs/java/memcache/

[18] Google. (2013, avgust) Task Queue Java API Overview. Dostopno na: https://developers.google.com/appengine/docs/java/taskqueue/

[19] Google. (2013, avgust) Users Java API Overview. Dostopno na: https://developers.google.com/appengine/docs/java/users/

[20] Google. (2013, avgust) Channel Java API Overview. Dostopno na: https://developers.google.com/appengine/docs/java/channel/

[21] The Eclipse Foundation. (2013, september) Eclipse IDE for Java EE Developers. Dostopno na: http://www.eclipse.org/downloads/moreinfo/jee.php

[22] T. E. Foundation. (2013, september) EGit. Dostopno na: http://www.eclipse.org/egit/

[23] Google. (2013, marec) Google Plugin for Eclipse. Dostopno na: https://developers.google.com/eclipse/

[24] (2013, september) About Moodle. Dostopno na: https://moodle.org/about/

[25] The jQuery Foundation. (2013, september) jQuery. Dostopno na: http://jquery.com/

[26] (2013, september) GitHub. Dostopno na: https://github.com/

[27] (2013, september) Bootstrap. Dostopno na: http://getbootstrap.com/2.3.2/

[28] Google. (2012, april) Google Charts. Dostopno na: https://developers.google.com/chart/

[29] T. Richardson. (2013, avgust) jQuery Timepicker Addon. Dostopno na: https://github.com/trentrichardson/jQuery-Timepicker-Addon

[30] Y. Fang. (2013, september) JSON Simple. Dostopno na: http://code.google.com/p/json-simple/

[31] O. Software. (2013, september) JPA 2 Annotations. Dostopno na: http://www.objectdb.com/api/java/jpa/annotations

[32] Google. (2013, avgust) Users Java API Overview. Dostopno na: https://developers.google.com/appengine/docs/java/users/

[33] R. W. Sebesta, *Programming the World Wide Web*, 7. izdaja. Addison-Wesley, marec 2012.

[34] S. Powers, *Learning JavaScript: Add Sparkle and Life to Your Web Pages.* O'Reilly Media, december 2008.

[35] J. Cooper. (2009, junij) Avoiding datastore contention. Dostopno na: https://developers.google.com/appengine/articles/scaling/contention

[36] Google. (2013, avgust) Visualization: Table. Dostopno na: https://developers.google.com/chart/interactive/docs/gallery/table

[37] Google. (2013, avgust) Google App Engine Billing FAQ. Dostopno na: https://developers.google.com/appengine/kb/billing

[38] Google. (2013, avgust) Quotas. Dostopno na: https://developers.google.com/appengine/docs/quotas

[39] Google. (2013, avgust) Java Datastore API. Dostopno na: https://developers.google.com/appengine/docs/java/datastore/