ივანე ჯავახიშვილის სახელობის
თბილისის სახელმწიფო
უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო
მეცნიერებათა ფაკულტეტი
კომპიუტერულ მეცნიერებათა
დეპარტამენტი

მიხეილ ჭელიძე

მარტივი CRUD აპლიკაციების გენერატორი

სამაგისტრო პროგრამა: ინფორმაციული სისტემები
სამაგისტრო ნაშრომი შესრულებულია ინფორმაციულ სისტემებში
მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი : **ტარიელ ხვედელიძე,**
თსუ ასოცირებული პროფესორი

თბილისი

2018

ანოტაცია

სამაგისტრო ნაშრომში განხილულია ინფორმაციული სისტემა, რომელიც ემსახურება გარკვეული სახის ინფორმაციის შეყვანას, ნახვას, რედაქტირებას და წაშლას.

სისტემა ორგანიზაციებში აგვარებს ისეთ ტიპიურ პრობლემას, როდესაც მათ არ აქვს საკმარისი დრო ან რესურსი, მაგრამ მონაცემების აღრიცხვა მეტ-ნაკლებად მნიშვნელოვანია.

განხილულია სისტემის მომხმარებელთა ტიპები. აღწერილია პროექტის არქიტექტურა და ტექნოლოგიები. განმარტებულია თუ რატომ იქნა შერჩებული ესა თუ ის მოდელი თუ ტექნოლოგია.

აქვე მინდა შევადარო სხვა სისტემებს (Google Forms, JotFoms) და აღვნიშნო დადებითი და უარყოფითი მხარეები.

უარყოფითი:

- არც თუ ისე დახვეწილია დიზაინის მხრივ.
- მოუხერხებელია მცირე ზომის ეკრანების მქონე აპარატურით მუშაობისას.
- ამ ეტაპზე არ აქვს სტატისტიკის დათვლის მოდული, რაც იგივე Google Forms-ში საკმაოდ კარგად არის გადაწყვეტილი.

დადებითი:

- შეიძლება სისტემის ისე დაყენება, რომ გარე სამყაროს შენს ინფორმაციაზე წვდომა არ ქონდეს. Google Formsის შემთხვევაში ეს შეუძლებელია, რადგან მონაცემებს ინახავ Googleს სერვერებზე.
- გააჩნია ჩადგმული ფორმები;
- აქვს შედარებით მოხერხებული კომონენტები.
- არის ვებ-აპლიკაცია, რაც პირველ რიგში გულისხმობს ოპერაცული სისტემისგან დამოუკიდებლობას, რაც დამეთანხმებით დიდი პლუსია.
- აქვს ცენტრალიზებული მონაცემთა შენახვის და დამუშავების სისტემა.
- არის მსუბუქი და არ საჭიროებს დიდ აპარატურულ რესურსებს, როგორც სერვერზე ასევე მომხმარებელთან.
- აქვს ავტომატური აუდიტ მხარდაჭერა

Anotation

In the following Master's work, informational system is reviewed as a possibility of creating, reading, updating and deleting some kind of information.

System has ability to reduce typical organizational problems, while the latter has no sufficient time and resource, but still has to carry on data management.

Presented work discusses types of system users, describes architectural decisions and responds on the following questions: What kind of technologies are used for system building and why have they been chosen?

Compared to other systems (Google Forms, JotForms), presented product has its advantages and disadvantages:

Disadvantages:

- Design isn't perfectly delicate;
- Creates uncomfortable experience while using small sized screen devices;
- At this stage, system does not have counting module for statistics, what we cannot say about Google Forms.

Advantages:

- System can be set in such manner that outside world may have no access on submitted information. Such possibility is not available for Google Forms, because all data is saved and uploaded on Google servers;
- Nested forms ability;
- Flexible components;
- Web application, which firstly means independence from operational system. Undoubtedly, this must be considered as a major advantage;
- Centralized system for saving and processing data;
- Soft System, which doesn't require a big Hardware Resource as well as on server, also with users;
- Automatic Audit Support.

სარჩევი

მარტივი CRUD აპლიკაციების გენერატორი

დღესდღეისობით ორგანიზაციებში მნიშვნელოვანია ინფორმაციის შეტანა, ნახვა, რედაქტირება, წაშლა. ორგანიზაციები ძირითადად ამ პრობლემის მოსაგვარებლად იყენებენ Microsoft Excel-ის ფაილებს, ხდება მოცემული ფაილების დროდადრო განახლება სხვადასხვა ადამიანის მიერ, რომელიც საბოლოო ჯამში ქმნის გაუგებრობას, ამას ემატება ისიც, რომ თუ სავალალო შემთხვევის წყალობით ეს ფაილები წაგვეშალა ან დაგვეკარგა შეიძლება საერთოდ ან ნაწილობრივ დავკარგოთ ინფორმაცია. ასევე მუშაობის ამ პრინციპს დამატებით ის მინუსი აქვს, რომ გამოსაყენებლად რთულია, უნდა გვქონდეს შესაბამისი პროგრამა რომელიც ყველა ოპერაციული სისტემისთვის არაა შექმნილი, ასევე თავს იჩენს ფაილების ვერსიების არათავსებადობა. ასევე მსგავსი მიდგომა საჭიროებს Excel-ს ფორმულების ცოდნას, ასავე საჭიროა გარკვეული დრო, რომ ადამიანმა ისწავლოს მასთან მუშაობა.

სწორედ მსგავსი პრობლემების ერთ-ერთი საუკეთესო გადაწყვეტაა სისტემის შექმნა, რომელიც ამ ყველა მონაცემებს ცენტრალიზირებულად მოუყრის თავს ერთ დაცულ სერვერზე. ნებისმიერი ადგილიდან, ნებისმიერ დროს შეგვიძლია მივწვდეთ სისტემას, რომელიც დაცულია და უზრუნველყოფს ჩვენი მონაცემების სიზუსტესა და სანდოობას. ასევე არ საჭიროებს სპეციფიურ ცოდნას და მარტივად შეუძლია ადამიანს აითვისოს დროის მცირე მონაკვეთში.

გარდა ამისა კიდევ მნიშვნელოვანია დრო, თუ რამდენ ხანში შეიძლება შეიქმნას აპლიკაცია. რეალობაში არის ისეთი შემთხვევები, რომ სასიცოცხლოდ მნიშვნელოვანია სწრაფად, მონაცმთა აღრიცხვის სისტემის შექმნა. ამიტომ დეველოპერებს უწევთ სწრაფად იმუშავონ, რაც იწვევს საბოლოო ჯამში უხარისხო პროდუქტის შექმნას, რომელიც ხშირ შემთხვევებში გადაკეთებას არ ექვემდებარება და საჭირო ხდება ხელახლა მისი დაწერა.

თავი1. სისტემის შესახებ

1.1. სისტემის კომპონენტები

სისტემა შეგვიძლია დავყოთ 2 ლოგიკურ კომონენტად:

- **Web API**
  ეს არის კოდის სერვერული ნაწილი რომელიც უშუალოდ იღებს და ამუშავებს სისტემაში შემოსულ მოთხოვნებს, აქ მუშავდება სისტემის ბიზნეს ლოგიკა, ეს კომპონენტი პასუხისმგებელია სისტემის გამართულად მუშაობაზე და შეცდომების დამუშავებაზე.

- **Web Client**
  ეს არის ის კოპონენტი სადაც მომხმარებლები მუშაობენ, და ვიზუალურად ხედავენ, და ბრძანებენს აძლებენ Web Apiს.

1.2. სისტემის მომხმარებლები

სისტემას ჰყავს 2 ძირითადი მომხმარებელი:

- **ადმინისტრატორი-ტექნიკური პირი.**
  ეს არის იმ ტიპის მომხმარებელი, რომელსაც შეუძლია აპლიკაციის მომხმარებელთა დამატება, როლების მინიჭება, აპლიკაციის დამატება. აპლიკაციების ობიექტებსი და ამ ობიექტებს შორის კავშირების განსაზღვრა.

- **ოპერატორი.**
  მას შეუძლია მონაცემების დამატება და მხოლოდ მისი დამატებული მონაცემების რედაქტირება.

1.3. არქიტექტურული გადაწყვეტები

სერვერის მხარე იქნება გამოყენებული Javას ერთ-ერთი გავრცელებული და პოპულარული Spring Framework და მისი რამდენიმე მოდული:

- **Spring Boot**
  ზედმეტი ძალისხმევის გარეშე შეგიძლია შექმნა დამოუკიდებელი აპლიკაცია, რომელიც დაფუძნებული იქნება Spring Frameworkის პროგრამულ მოდელზე.

- **Spring Web**
  გაძლევს შესაძლებლობას მარტივად გადააქციო შენი აპლიკაცია ვებ-აპლიკაციად და გავიწყებს იმ სირთულეებს რაც ვებ აპლიკაციის სერვერულ მხარეს შეიძლება ჰქონდეს.

- **Spring Data Jpa**
  Spring Framework გაძლევს საშუალებას გამოიყენო JPA და მისი რომელიმე პროვაიდერი.

- **Spring Data**
  ამ მოდულით გეძლევა საშუალება წვდომა მონაცემებზე, როგორც რელაციურ ასევე არარელაციურ ბაზებში.

- **Spring security**

ცხრილების აუდიტინგისთვის გამოყენებულია Hibernate Envers. Hibernate Envers იძლევა საშუალებას მატივად ვაწარმოვოთ აუდიტირება და ვერსიების შენახვა მონაცემებზე.

ბაზის მხარეს გამოყენებულია რელაციური ბაზა: Postgres;

მომხმარებლის (კლიენტის) მხარეს არის გამოყენებლი ExtJs 6
ExtJS გეხმარება შექმნა ჭარბ მონაცემებიანი მულტიპლატფორმული ვებ აპლიკაცია.

თავი 2. სამომხმარებლო ინტერფეისის აღწერა

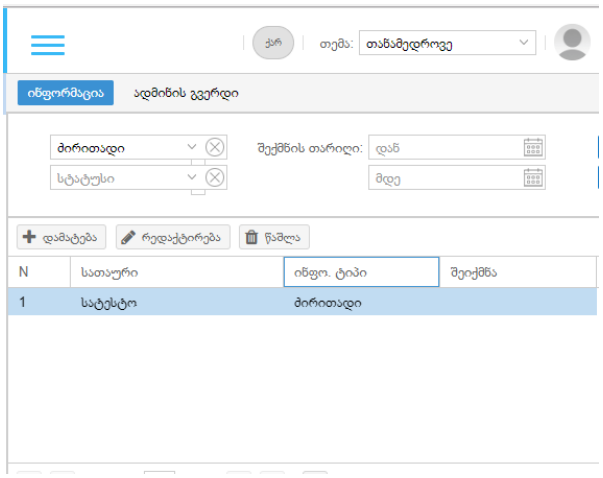
2.1. პროგრამის ზოგადი სტრუქტურა

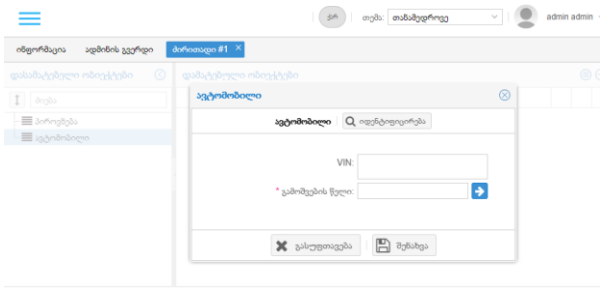
როგორც უკვე ავღნიშნე პროგრამას ჰყავს ორი ტიპის მომხმარებელი: ადმინისტრატორი და ოპერატორი. ადმინისტრატორის ფუნქციის მქონე მომხმარებლით ავტორიზაციისას ჩვენ დავინახავთ ორ ჩანართს (tab), “ინფორმაცია”, “ადმინის გვერდი”. ოპერატორ მომხმარებელს აქვს მხოლოდ ერთი ჩანართი “ინფორმაცია”.

2.1.1. ინფორმაცია

ჩანართში, მომხმარებელს საშუალება აქვს შეიტანოს, მოძებნოს, დაარედაქტორის ან წაშალოს ინფორმაცია. მომხმარებელს ასევე საშუალება აქვს მონაცემები დაახარისხოს “ფოლდერებში”.

ინფორმაციის შეტანისთვის მომხმარებელი პირველ რიგში ქმნის “ფოლდერს” ან არსებულს ხსნის. და იქ ხვდება შესატანი ფორმები. ამოირჩევს სასურველ ფორმას, შეავსებს ველებს, და შეინახავს. კონკრეტული შევსებული ფორმის მოსაძებნად მომხმარებელი ხსნის ფორმას, და აწვება ინდენტიფიცირება ღილაკს. მას საშაუალება აქვს მოძებნოს, ყველა ველის მიხედით. თუ ველი რიცხვითია, ან თარიღის ტიპის, შესაძლებელია შუალედების მითითებაც.





აქვე მინდა მოვიყვანო ძებმის მეთოდი

```
private AbstractQuery<InfoInsertObject>
findInfoInsertObjects(InfoInsertQueryModel queryModel, CriteriaBuilder cb,
AbstractQuery<InfoInsertObject> query) throws ParseException,
IllegalAccessException, InvocationTargetException, IntrospectionException,
InstantiationException, InfoInsertConvertException, NoSuchMethodException {
    if (queryModel.getProperties() == null || queryModel.getProperties().isEmpty())
return null;

    if (cb == null) cb = em.getCriteriaBuilder();

    boolean isSubQuery = true;
    if (query == null) {
        query = cb.createQuery(InfoInsertObject.class);
        isSubQuery = false;
    }
    Root<InfoInsertObject> table = query.from(InfoInsertObject.class);
    if (!isSubQuery) {
        ((CriteriaQuery) query).select(table);
    } else {
        ((Subquery) query).select(table);
    }


    Collection<Predicate> predicates = new ArrayList<>();

    Long objectTypeId = queryModel.getObjectTypeId();
    if (objectTypeId != null && objectTypeId > 0) {
        predicates.add(cb.equal(table.get("type").get("id"), objectTypeId));
    }
    List<InfoInsertQueryValueModel> values = queryModel.getProperties();

    if (values != null && !values.isEmpty()) {
```

```
        Join<InfoInsertObject, InfoInsertObjectValue> objectValueJoin =
table.join("values", JoinType.LEFT);
        Collection<Predicate> valuePredicate = new ArrayList<>();
        for (InfoInsertQueryValueModel valueModel : values) {

            Predicate propertyIdPredicate =
cb.equal(objectValueJoin.get("property").get("id"), valueModel.getPropertyId());
            Predicate propertyValuePredicate = null;

            Object value = valueModel.getSingleValueByValueType();
            switch (valueModel.getValueType()) {
              case LIBRARY:
                if (value instanceof String || value instanceof Number)
                  propertyValuePredicate =
objectValueJoin.get("valueLibrary").get("id").in(hierarchyOfLibraryValue(Long.value
Of(value.toString())));
                  propertyValuePredicate =
cb.equal(objectValueJoin.get("valueLibrary").get("id"),
Long.valueOf(value.toString()));
                break;
              case BOOLEAN:
                if (value instanceof String || value instanceof Boolean)
                  propertyValuePredicate =
cb.equal(objectValueJoin.get("valueBoolean"), Boolean.valueOf(value.toString()));
                break;
              case NUMBER:
                if (valueModel.isRange()) {
                  propertyValuePredicate = cb.or(
                      getRangePredicate(cb, objectValueJoin.get("valueNumber"),
valueModel.getFromTo()),
                      getRangePredicate(cb, objectValueJoin.get("valueNumberTo"),
valueModel.getFromTo())
                  );
                } else if (value != null) {
                  propertyValuePredicate = cb.or(
```

```
                        cb.equal(objectValueJoin.get("valueNumber"),
Double.valueOf(value.toString())),
                        cb.equal(objectValueJoin.get("valueNumberTo"),
Double.valueOf(value.toString()))
                    );
                }
                break;
            case STRING:
                if (value != null) {
                    Join valueStringJoin = objectValueJoin.join("valueString",
JoinType.LEFT);
                    propertyValuePredicate = cb.like(valueStringJoin.get("value"), '%' +
((String) value).trim() + '%');
                }
                break;
            case DATE:
                if (valueModel.isRange()) {
                    propertyValuePredicate = cb.or(
                        getRangePredicate(cb, objectValueJoin.get("valueDate"),
valueModel.getFromTo()),
                        getRangePredicate(cb, objectValueJoin.get("valueDateTo"),
valueModel.getFromTo())
                    );
                } else if (value != null) {
                    propertyValuePredicate = cb.or(
                        cb.equal(objectValueJoin.get("valueDate"), value),
                        cb.equal(objectValueJoin.get("valueDateTo"), value)
                    );
                }
                break;
            case OTHER_OBJECT:
                value = valueModel.getValue();
                if (value instanceof InfoInsertQueryModel) {
                    Subquery<InfoInsertObject> subquery =
query.subquery(InfoInsertObject.class);
```

```
                    subquery = (Subquery<InfoInsertObject>)
findInfoInsertObjects((InfoInsertQueryModel) value, cb, subquery);

                    propertyValuePredicate =
objectValueJoin.get("valueInfoInsertObject").in(subquery);
                }
                break;
            case OTHER_SOURCE:
                if (value instanceof Long) {
                    Join<InfoInsertObjectValue, InfoInsertValueOtherSource>
valueOSJoin = objectValueJoin.join("valueOtherSource", JoinType.LEFT);
                    propertyValuePredicate =
cb.equal(valueOSJoin.get("osObjectType").get("id"), value);
                }
                break;
        }
        if (propertyIdPredicate != null && propertyValuePredicate != null) {
            // line from query: 10-11; 13-14 (see comment on method)
            valuePredicate.add(cb.and(propertyIdPredicate, propertyValuePredicate));
        }
    }
    if (!valuePredicate.isEmpty()) {
        query.groupBy(table.get("id"), table.get("type"), table.get("rating"));
        // line from query: 18 (see comment on method)
        query.having(cb.and(cb.greaterThanOrEqualTo(cb.count(table), (long)
valuePredicate.size()), cb.equal(cb.countDistinct(table.get("id")), 1L)));
        predicates.add(cb.or(valuePredicate.toArray(new
Predicate[valuePredicate.size()])));
    } else {
        return null;
    }
}


if (!predicates.isEmpty()) {
```

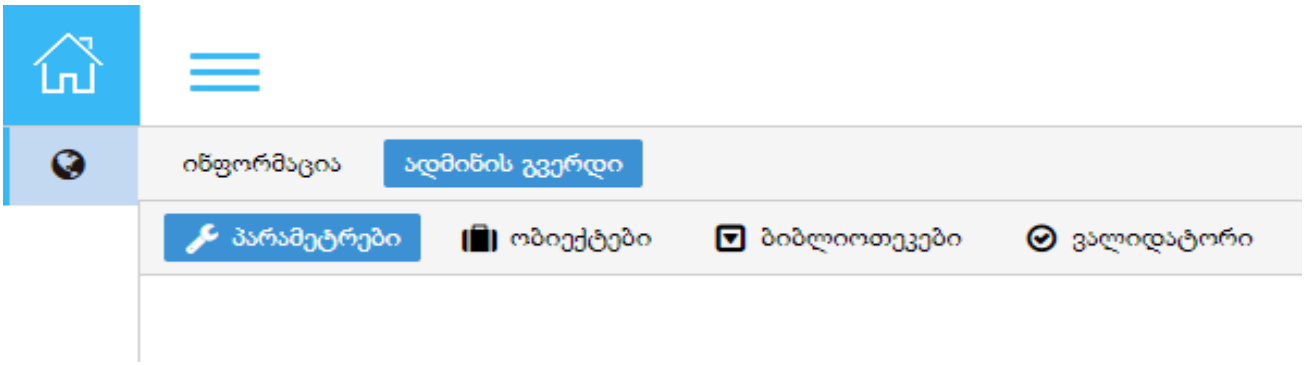
```
            query.where(predicates.toArray(new Predicate[predicates.size()]));
        }

        return query;

    }
```

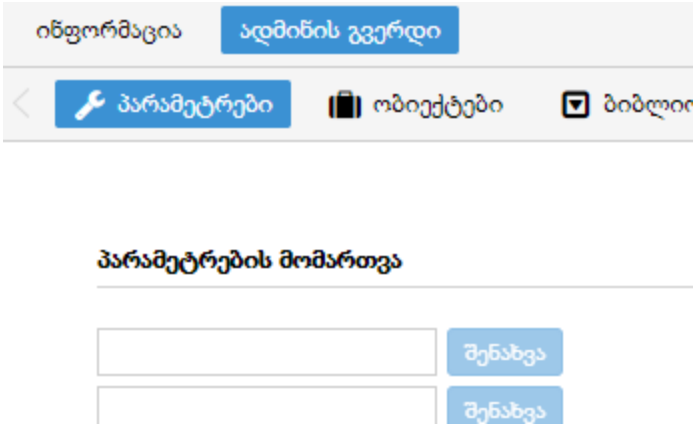
2.1.2 . ადმინის გვერდი

ჩანართში, ადმინისტრატორ-ტექნიკურ პირს კიდევ დამატებით რამდენიმე ჩანართი უჩანს.



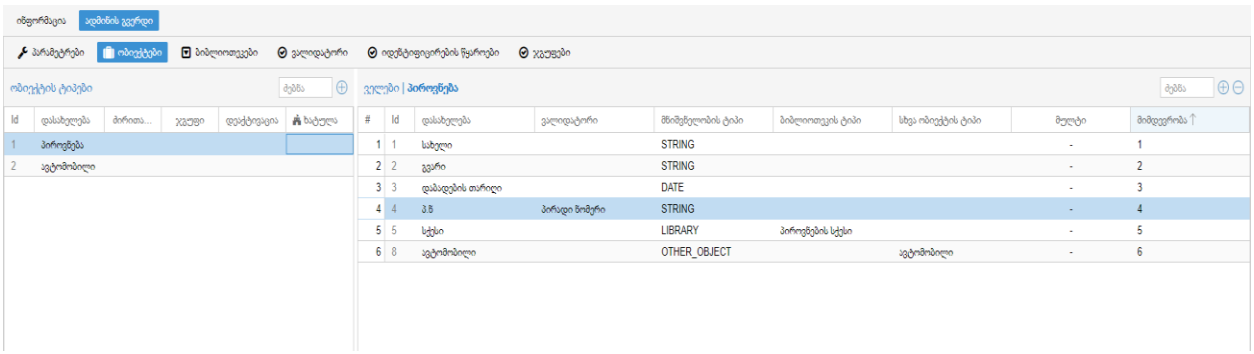
2.1.2.1. პარამეტრები

აქ კოდში ჩარევით შეუძლია დამატებითი ფუნქციონალი შესძინოს აპლიკაციას.



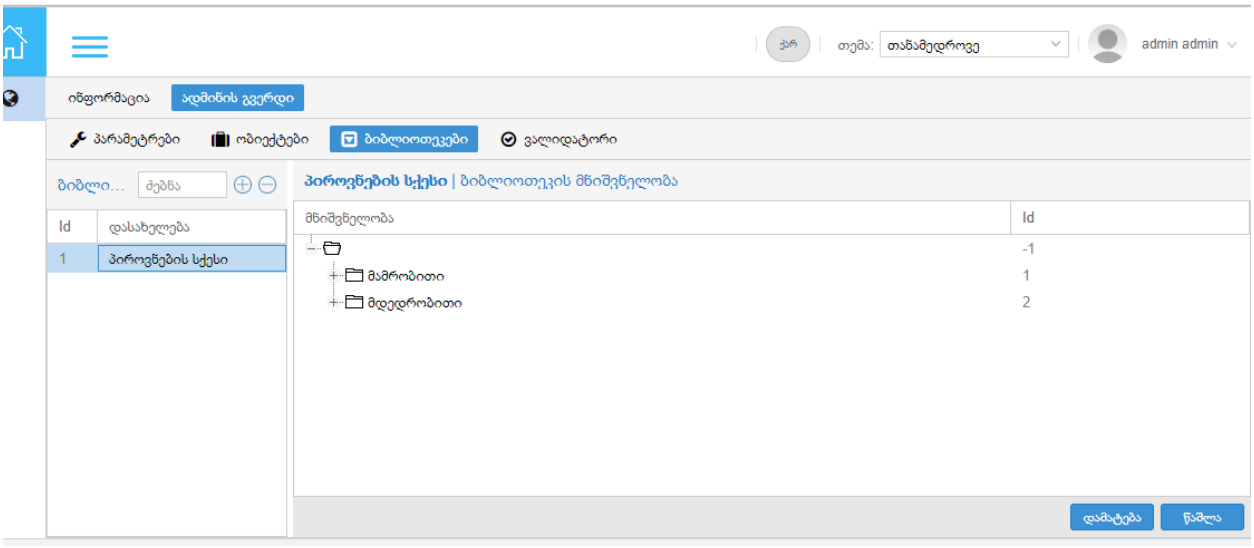
2.1.2.2. ობიექტები

აქ მომხმარებელი განსაზღვრავს შესატან ველებს. ახდენს მათ კატეგორიზაციას ობიექტებში. ველებს უწერს შესაბამის დასახელებას, მნიშვნელობის ტიპს, ადებს შეზღუდვებს (ვალიდატორებს), განსაღვრავს ველი ერთ მნიშვნელობას შეინახავს თუ მრავალს და ა.შ.



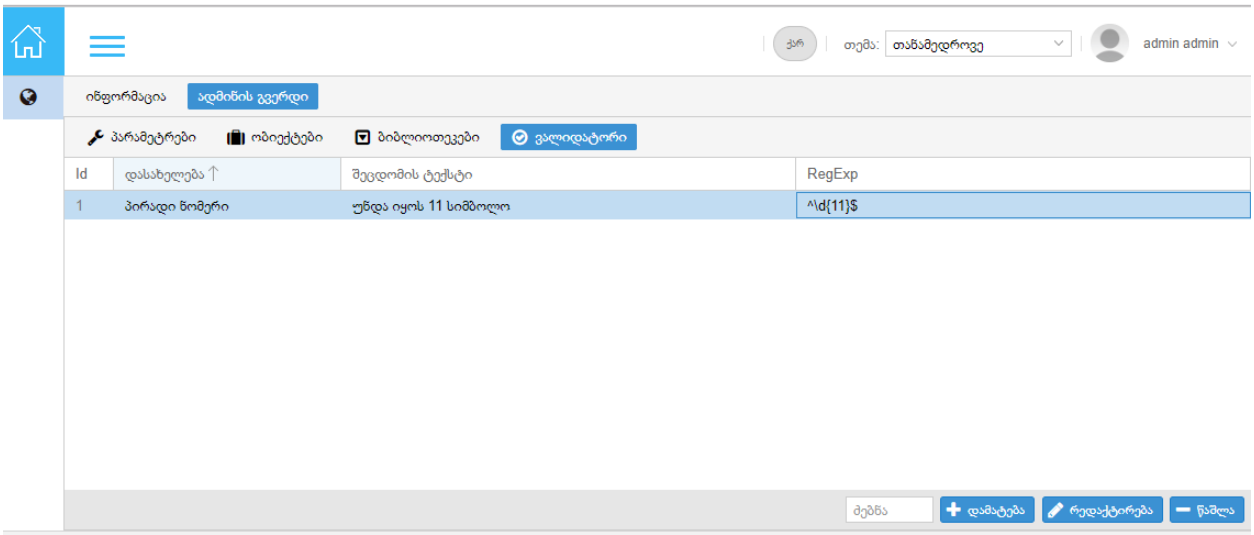
2.1.2.3. ბიბლიოთეკები

აქ ადმინისტრატორს შესაძლებლობა აქვს შექმნას ამოსარჩევი მნიშვნელობების ჩამონათვალი. როგორიცაა მაგალითად სქესი, ქვეყნების ჩამონათვალი, ავტომობილის მარკა-მოდელი და ა.შ.



2.1.2.4. ვალიდატორი

როგორც ზემოთ ავღნიშნე ადმინისტრატორს შუძლია ზოგიერთ ველს დაადოს შეზღუდვა, არ მისცეს ოპერატორ მომხმარებელს ყველანაირი მნიშვნელობის ჩაწერის საშუალება. მაგალითისათვის ავიღოთ ველი "ელექტორნული ფოსტა", როგორც ჩვენთვის ცნობილია, ფოსტის მისამართს გარკვეული ფორმა აქვს. ამ ველის შევსებისას არ უნდა შეიძლებოდეს abcabc-ს ჩაწერა. ამიტომ ადმინისტრატორი შესაბამის RegEx ვალიდატოს დაწერს და დაადებს ველს შეზღუდვად.



დასკვნა

ნაშრომში განხილული სისტემა, როგორც ანოტაციაში ავღნიშნე, არაა პიონერი თავის ჭრილში. მის კონკურენრად შეიძლება ჩაითვალოს Google Forms, JotForms. აღნიშნული აპლიკაციებიც ემსახურება გარკვეულწილად მონაცემების დაგროვებას. თუმც ჩემს აპლიკაციასას მათთან შედარებით აქვს როგორც გარკვეული უპირატესობები, ასევე მსგავსი თავისებურებები და , როგორც ვიცით უნაკლო არაფერია - აქვს თავისი მინუსებიც.

პირველ რიგში მინდა დავიწყო მინუსებით. აპლიკაციას შეიძლება დავუწუნოთ დიზაინი. ალბათ ეს გასაგები ხარვეზია, იმის გათვალისწინებით რომ ზემოაღნიშნული კომპანიები საკმაოდ დიდ მატერიალურ რესურს ხარჯავენ დიზაინის და სამომხმარებლო ინტერფეისის ინტუიტიურობის განვითარბაზე. თუმცა აქვე მინდა ავღნიშნო, რომ ჩემს აპლიკაციაში, მათი გამოცდილება ამ კუთხით მეტ-ნაკლებად გამოყენებულია.

ასევე დიზაინის ხარვეზად შეიძლება ჩაითვალოს ის ფაქტიც რომ აპლიკაცია მოუხერხებელია მცირე ზომის ეკრანების მქონე აპარატურით მუშაობისას, მე მაინც მინდა ეს უკანასკნელი ცალკე პუნქტად გავიტანო. თუმც ამის საპირწონედაც, მსურს დავამატო, რომ მე ძირითადად ე.წ. უკანა მხირს დეველოპერი (backend) ვარ, და დიზაინზე მუშაობა არამარტო დიდ სიამოვნებას არ მანიჭებს, არამედ, შეიძლება ითვას ამ საკითხში საკმაოდ სუსტი გახლავართ.

Google Forms-ისგან განსხვავებით, აპლიკაციას ამ ეტაპზე არ გააჩნია სტატისტიკის მოდული. აპლიკაციის სახელიდან გამომდინარე, მარტივი CRUD აპლიკაციების გენერატორი, შეიძლება ითქვას, რომ არც სჭირდება სტატისტიკა. მაგრამ ჩემი გადმოსახედიდან, ამ აპლიკაციის მაშტაბურობას თუ გავითვალისწინებთ, ძალიან ცოტა შემთხვევა შეიძლება მოვიფიქროთ, როდესაც სტატისტიკა შეიძლება არ იყოს საჭირო.

რაც შეეხება დადებით მხარეებს, თვალში საცემია ის ფაქტი, რომ არი ვებ აპლიკაცია. რაც თავის თავში გულისხმობს იმას რომ დამოუკიდებელია ოპერაციული სისტემისგან, არ მოითხოვს მომხმარებლისგან დამატებითი აპლიკაციების დაყენებას (ყველა თავმოყვარე იპერაციულ სისტემას ერთი მაინც ბრაუზერი მოყვება.)

არც Google Forms და არც JotForms არ აქვს ჩადგმული ფორმები. თუ ფორმას ერთ ობიექტად წარმოვიდგენთ და მასში ზოგიერთ ველს დავაჯგუფებთ რაიმე ნიშნით და მეორე ობიექტად გამოვსახავთ, შეიძლება ვთვათ რომ ერთ ფორმაში მეორე ფორმაა. ორ ფორმად აღქმა არ დაგვჭირდებოდა, თუ არ გავითვალისწინებდით ერთ გარემოებას: შესაძლებელია უპრიანი ყოფილიყო ერთ ე.წ. ჩადგმული ფორმის რამდენჯერმე შევსება დიდ ფორმაში. ასეთ დროს ჩადგმულობა არის ერთ-ერთი საინტერესო გამოსავალი. (მაგალითსთვის: პიროვნებას ვავსებთ თავისი ავტომობილებით. ყველასთვის ცნობილია რომ პიროვნებას შეუძლია ქონდეს რამდენიმე ავტომობილი. ერთ ფორმაში რამდენიმე ავსებს ავტომობილის რამდენიმე ფორმას და პიროვნების ერთს.)

ველებს რაც შეეხება, გვაქვს საშუალება გვქონდეს რამდენიმე მნიშვნელობიანი ველი. თარიღისა და რიცხვით მნიშვნელობებში შესაძლებელია შუალედების შეტანაც.

ხშირად არის ხოლმე, რომ კომპანიას სჭირდება ისტორია მონაცემების ცვლილვების შესახებ. არსებობს ზოგჯერ იმის საფრთხე, რომ მომხმარებელმა შემთხვევით მონაცემი წაშალოს. ამქედან გამომდინარე სისტემა აკეთებს მონაცემების აუდიტინგს. მოაცამების ასლები ინახება სხვა ცხრილებში. ინახება ჩანაწეის ცვლილების თარიღი და რა ტიპის ცვლილება მოხდა (შეიქნმა, დარედაქტირდა, წაიშალა).

ასევე საყურადღებოა ის, რომ სისტემა არაა რთულად გამოსაყენებლი და გასამართი. ადვილია მისი ფუნქციონალის ოპერატორიებისთვის ახსნა. მარტივად ხდება ფორმების აწყობაც. აპლიკაცია არის მოქნილი და გათვალისწინებულია ბევრი ნიუანსი, რაც შეიძლება მისი ექსპლუატაციის პერიოდში მომხმარებელს დასჭირდეს.

გამოყენებული ლიტერატურა

1. Spring In Action - http://www.manning.com/walls5/
2. Spring MVC - http://www.apress.com/9781430241553
3. ExtJS - https://docs.sencha.com/extjs/6.0.0/index.html
4. Hibernate Envers - http://hibernate.org/orm/envers/