ARTIGO: 11375
Iniciando uma aplicação Android
Introdução

Nesse artigo veremos como configurar e estruturar uma aplicação Android trabalhando com o Veloster Framework. Vamos fazer uma aplicação simples, que deve:

- Armazenar o nome e o telefone de um contato
- Pesquisar contatos pelo nome

Maven

Primeiramente temos que configurar uma nova aplicação android, e para isso vamos utilizar o maven. No pom.xml, temos que adicionar as dependências do Veloster

XML Code

```
<!-- download artifacts from this repo -->
<repositories>
    <repository>
        <id>mobile-mind-nexus</id>
        <name>vineetmanohar</name>
        <url>http://nexus.mobilemind.com.br/nexus/content/repositories/mobile-mind</url>
        <releases>
            <enabled>true</enabled>
        </releases>

        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
    <repository>
        <id>mobile-mind-droid-nexus</id>
        <name>vineetmanohar</name>
        <url>http://nexus.mobilemind.com.br/nexus/content/repositories/mobile-mind-droid</url>
        <releases>
            <enabled>true</enabled>
        </releases>

        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>

<dependencies>
  <dependency>
      <groupId>br.com.mobilemind.api</groupId>
      <artifactId>veloster-droid</artifactId>
      <version>1.14</version>
  </dependency>
  <dependency>
      <groupId>br.com.mobilemind.api</groupId>
      <artifactId>veloster-api</artifactId>
      <version>1.14</version>
  </dependency>
</dependencies>
```

Configuração

Agora precisamos definir os arquivos de configuração do Veloster. Vamos criar os arquivos de configurações e mensagens de validação.

file: resources.properties

Java Code

```
br.com.mobilemind.db.android=true
br.com.mobilemind.db.driver=
br.com.mobilemind.db.name=contacts.db
br.com.mobilemind.db.testName=contacts_test.db
br.com.mobilemind.db.path=
br.com.mobilemind.db.pathEnv=
br.com.mobilemind.db.host=
br.com.mobilemind.db.port=
br.com.mobilemind.db.backupPath=contacts
br.com.mobilemind.db.user=
br.com.mobilemind.db.password=
```

```
#ddl can be create | update | none
br.com.mobilemind.db.ddl=update
br.com.mobilemind.android.applicationPakage=br.com.velster.contacts
br.com.mobilemind.android.dataBaseVersion=1
br.com.mobilemind.android.db.Location=/data/{0}/databases/{1}
br.com.mobilemind.db.backupSufixFormat=dd-MM-yyyy_HH-mm-ss
br.com.mobilemind.defaultDateFormat=yyyy-MM-dd hh:mm:ss.SSS
```

Podemos reparar que estamos utilizando a configuração *br.com.mobilemind.db.ddl=update* definindo que os campos e tabelas serão criadas automáticamentes pelo Framework.

Validações

file: message_validates.properties

Java Code

```
br.com.mobilemind.date=date range should be between {0} and {1} for field {2}
br.com.mobilemind.notnull=value cannot be null for field {0}
br.com.mobilemind.length=length range should be between {0} and {1} for field {2}
br.com.mobilemind.number=number range should be between {0} and {1} for field {2}
br.com.mobilemind.regex=invalid value. required format {0} for field {1}
br.com.mobilemind.sdcardNotAvailable=sdcart not is available
br.com.mobilemind.sdcartNotCanWrite=fail on try writer in sdcard
```

Agora, na inicialização de nossa aplicação, vamos configurar o uso do Veloster:

Java Code

```
VelosterDroid.build(getApplication());
```

Entidades

Vamos criar as entidades da aplicação.

Java Code

```
public classe Contact extends EntityLazy<Contact>{
    @Id()
    @Column()
    private Long id;

    @Length(min=2, max=100)
    @NotNull
    @Column
    String name

    @Length(min=10, max=15)
    @NotNull
    @Column
    String phone

   // getters and setters
}
```

Controladores

Agora temos que criar nosso repositório para a classe *Contact*

Java Code

```
public class ContactRepository{
    private Veloster<Contact> veloster;

    public ContactRepository(){
        veloster= VelosterRepository.getVeloster(Contact.class);
    }

    public void save(Contact entity){
      veloster.save(entity);
    }

    public void deleteContact entity){
      veloster.delete(entity);
    }

    public void update(Contact entity){
      veloster.update(entity);
    }

    public List<Contact> listAll(){
      return veloster.list();
    }

    public Contact findById(Long id){
      return veloster.load(id);
    }
```

```
    public List<Contact> findByStartsName(String name){
      Criteria<Contact> criteria = veloster.createCriteria();
      return criteria.add("name", new Like(name, Match.STARTWITH)).list()
    }
}
```

Ou como alternativa, podemos criar

Java Code

```
public interface ContactRepository extends RepositoryModel<Contact>{

   List<Contact> findAllByName(String name);
}
```

View

E por fim, já vamos criar nossas classes que manipulam as views do Android. A única coisa que deve-se ter cuídado é quanto ao tratamanto de exceções. Por exemplo, em um método save:

Java Code

```
public void save(){
   final ContactRepository repository = new ContactRepository();

   try{
      repository.save(getEntity());
   }catch(EntityValidator e){
    //display validation alert message
   }catch(VelosterException e){
    // display fatal error message
   }
}
```

Conclusão

E com isso podemos ver como é simples configurar o Veloster Framework em uma aplicação Android e usar todos seu recursos.