

دایکومنت پروژه پیاده‌سازی DSL برای مدیریت فروشگاه آنلاین با استفاده از ANTLR

فهرست

| | |
|----|--|
| ۱ | مقدمه |
| ۲ | کاربرد پروژه و مشکلاتی که حل می‌کند |
| ۳ | ویژگی‌ها و عملکردها |
| ۴ | مراحل پیاده‌سازی |
| ۵ | گرامر زبان و نحوه عملکرد |
| ۱۰ | توضیح یک مثال مربوط به ورودی و خروجی پروژه |
| ۱۰ | نمونه ورودی |
| ۱۲ | تولید کد برای ورودی مثال زده شده |
| ۱۷ | منابع و ابزارهای استفاده شده |

مقدمه

این پروژه به پیاده‌سازی یک زبان خاص دامنه (DSL) برای مدیریت فروشگاه آنلاین پرداخته است. زبان DSL طراحی شده با استفاده از ANTLR نسخه ۴ پیاده‌سازی شده و به مدیران فروشگاه آنلاین این امکان را می‌دهد که به صورت ساده و مؤثر اقدام به انجام عملیات مختلفی مانند ایجاد، حذف، به‌روزرسانی و لیست کردن موجودیت‌ها (مانند محصولات، کاربران و غیره) کنند. این زبان همچنین امکان اعمال تخفیف بر موجودیت‌ها و مدیریت ویژگی‌های آن‌ها را فراهم می‌کند.

در این زبان، چندین دستور اصلی وجود دارد که هر کدام عملکرد خاصی را انجام می‌دهند. دستورات شامل **define** برای تعریف موجودیت‌ها، **create** برای ایجاد موجودیت‌های جدید، **delete** برای حذف موجودیت‌ها، **update** برای به‌روزرسانی ویژگی‌های موجودیت‌ها، **list** برای مشاهده موجودیت‌ها، و **discount** برای اعمال تخفیف به موجودیت‌ها هستند.

هر دستور از یک ساختار خاص پیروی می‌کند که شامل مشخص کردن نوع موجودیت، ویژگی‌ها و مقادیر مربوطه است. این گرامر به گونه‌ای طراحی شده است که هر دستور به راحتی توسط سیستم تجزیه می‌شود و عملیات مربوط به آن را در سیستم فروشگاه آنلاین اجرا می‌کند.

کاربرد پروژه و مشکلاتی که حل می کند

مدیریت فروشگاه آنلاین معمولاً شامل وظایف پیچیده و تکراری مانند ایجاد، حذف، به روزرسانی محصولات، مدیریت کاربران و اعمال تخفیف ها است. این فرآیندها اغلب زمان بر، مستعد خطا و نیازمند دانش فنی هستند.

مشکلات موجود:

۱. پیچیدگی کدنویسی برای انجام عملیات ساده.
۲. رابط های کاربری گرافیکی سخت و زمان بر.
۳. محدودیت در سفارشی سازی و گسترش عملیات.

راه حل ارائه شده:

زبان خاص دامنه (DSL) طراحی شده در این پروژه این مشکلات را با رویکردی ساده و خوانا حل می کند:

- سادگی: دستورات ساده و خوانا برای مدیریت عملیات فروشگاه.
- کاهش خطا: شناسایی دستورات نادرست در مرحله تجزیه.
- سرعت بالا: انجام سریع وظایف تکراری با استفاده از دستورات کوتاه.
- قابلیت توسعه: امکان گسترش زبان برای نیازهای خاص هر فروشگاه.

نمونه کاربرد:

به جای کدنویسی پیچیده، کاربران می توانند با دستوری مانند زیر یک محصول جدید اضافه کنند:

```
} create
```

```
,entity: product
```

```
attributes: { name: "Laptop", price: 1200, stock: 50 }
```

```
{
```

این پروژه راه حل مؤثری برای ساده سازی مدیریت فروشگاه آنلاین است و نیاز به ابزارهای پیچیده یا دانش فنی بالا را کاهش می دهد.

ویژگی‌ها و عملکردها

زبان خاص دامنه (DSL) طراحی شده برای مدیریت فروشگاه آنلاین شامل ویژگی‌های زیر است:

۱. تعریف موجودیت‌ها:

- این ویژگی امکان تعریف موجودیت‌های جدید مانند سفارش (Order) و مشتری (Customer) را با مجموعه‌ای از ویژگی‌ها فراهم می‌کند.

۲. ایجاد موجودیت‌ها:

- این ویژگی به کاربران اجازه می‌دهد که موجودیت‌های تعریف‌شده را با مقادیر دلخواه در سیستم ایجاد کنند.

۳. به‌روزرسانی موجودیت‌ها:

- با استفاده از این ویژگی، کاربران می‌توانند اطلاعات موجودیت‌های موجود را تغییر دهند و مقادیر جدیدی به ویژگی‌های آن‌ها اختصاص دهند.

۴. حذف موجودیت‌ها:

- این ویژگی امکان حذف موجودیت‌ها از سیستم را با استفاده از یک شناسه منحصر به فرد فراهم می‌کند.

۵. مشاهده لیست موجودیت‌ها:

- این قابلیت به کاربران اجازه می‌دهد تا لیستی از تمام موجودیت‌های مشخص شده را مشاهده کنند.

مراحل پیاده‌سازی

۱. طراحی گرامر DSL:

- گرامر زبان با استفاده از ANTLR طراحی شد تا بتوان دستورات مربوط به مدیریت فروشگاه را تعریف کرد.
- گرامر شامل دستورات اصلی مانند create, delete, update, list و discount است.

۲. ایجاد Parser و Lexer:

- با استفاده از ANTLR، Parser و Lexer تولید شدند که مسئول تجزیه دستورات ورودی و تولید ساختارهای نحوی مربوطه هستند.

۳. ساخت درخت نحوی انتزاعی (AST):

- دستورات ورودی پس از تجزیه، به درخت نحوی انتزاعی تبدیل می‌شوند که نمایانگر ساختار منطقی دستورات است.

۴. تفسیر دستورات:

- دستورات از درخت نحوی خوانده و به عملیات قابل اجرا در سیستم تبدیل می‌شوند. به عنوان مثال:
- دستور create محصولی جدید در پایگاه داده ثبت می‌کند.
- دستور update ویژگی‌های محصول موجود را تغییر می‌دهد.

۵. آزمایش و اصلاحات:

- دستورات مختلف DSL برای اطمینان از صحت و عملکرد صحیح مورد آزمایش قرار گرفتند.
- اصلاحات لازم برای شناسایی خطاهای ورودی و بهبود خوانایی انجام شد.

گرامر زبان و نحوه عملکرد

۱. بخش ابتدایی گرامر: start و program

```
grammar StoreDSL;  
  
start: program EOF;  
  
program: statement*;
```

- start: این گرامر با تولید یک نود از نوع program که شامل تعدادی از دستورها (statement) است، شروع می‌شود. EOF (End of File) نشان می‌دهد که ورودی باید در انتها تمام شود.

- program: یک برنامه شامل تعداد نامحدودی از دستورها است که با statement بیان می‌شود. این یعنی می‌توان تعداد زیادی دستور داشته باشیم یا اصلاً هیچ دستور نداشته باشیم.

۲. statement

```
statement  
: define  
| create  
| delete  
| update  
| list  
| discount  
;
```

- statement: این بخش از گرامر، تمام انواع دستورات قابل پذیرش در زبان DSL را مشخص می‌کند. دستورات مختلفی مانند define, create, delete, update, list و discount وجود دارند که هر کدام به صورت جداگانه تعریف شده‌اند.

- هر دستور به صورت یک alternative یا گزینه (از طریق علامت |) در نظر گرفته شده است. این نشان‌دهنده این است که دستور ورودی باید یکی از این نوع‌ها باشد.

۳. دستور define

```
define
  : 'define' '{'
    'entity' ':' entityName ','
    'attributes' '{' attribute (',' attribute)* '}'
  '}';
```

- define: این دستور برای تعریف یک موجودیت (entity) جدید استفاده می‌شود. در داخل بخش define:
- 'define' کلمه کلیدی برای شروع دستور است.
- entity که به یک نام موجودیت اختصاص داده شده است (entityName).
- attributes که یک مجموعه از ویژگی‌ها (attributes) است. هر ویژگی شامل یک attribute می‌باشد و می‌توان به تعداد دلخواه ویژگی‌ها را اضافه کرد.
- تمام این اطلاعات در داخل یک بلوک {} قرار دارند.

۴. دستور create

```
create
  : 'create' '{'
    'entity' ':' entityName ','
    'attributes' '{' keyValuePair (',' keyValuePair)* '}'
  '}';
```

- create: این دستور برای ایجاد یک موجودیت جدید (مانند یک محصول یا کاربر) در فروشگاه آنلاین استفاده می‌شود. مشابه با دستور define، اما در اینجا ویژگی‌ها به صورت keyValuePair که شامل یک کلید و مقدار است، وارد می‌شوند.
- keyValuePair برای هر ویژگی به شکل یک جفت کلید و مقدار است.

۵. دستور delete

```
delete
  : 'delete' '{'
    'entity' ':' entityName ','
    'attributes' '{' keyValuePair '}'
  '}';
```

- delete: این دستور برای حذف موجودیت‌های خاص از سیستم است. در اینجا نیز مشابه دستورات قبلی، entity مشخص می‌کند که کدام موجودیت حذف شود، و ویژگی‌ها به صورت keyValuePair آمده است. در اینجا برای عملیات delete از id استفاده می‌شود.

۶. دستور update

```
update
  : 'update' '{'
    'entity' ':' entityName ','
    'attributes' '{' keyValuePair (',' keyValuePair)* '}'
  '}';
```

- update: این دستور برای به‌روزرسانی ویژگی‌های یک موجودیت استفاده می‌شود. دستورات keyValuePair مشابه با دستور create در اینجا نیز برای به‌روزرسانی اطلاعات موجودیت استفاده می‌شود.

۷. دستور list

```
list
  : 'list' '{'
    'entity' ':' entityName
  '}';
```

- list: این دستور برای نمایش فهرستی از موجودیت‌ها است. فقط به نام موجودیت نیاز است و اطلاعات اضافی وارد نمی‌شود.

۸. تعریف name و entityName

```
name
  : ID;

type
  : 'String' | 'Integer' | 'Float';

value
  : STRING | INT | FLOAT;

entityName
  : ID;
```

- name: این قانون نشان می‌دهد که نام‌ها در زبان باید به صورت شناسه (ID) تعریف شوند.

- entityName: مشابه name است و معمولاً برای موجودیت‌ها استفاده می‌شود.

- ID: شناسه‌هایی هستند که با یک حرف (LETTER) شروع می‌شوند و می‌توانند شامل حروف، اعداد، یا کاراکتر _ باشند.

۹. نوع داده‌ها: type

```
type
  : 'String' | 'Integer' | 'Float';
```

'String' | 'Integer' | 'Float'

- این قانون سه نوع داده‌ی اصلی را مشخص می‌کند:

- String: رشته‌ای از کاراکترها.

- Integer: عدد صحیح.

- Float: عدد اعشاری.

- این نوع داده‌ها معمولاً برای تعریف ویژگی‌های موجودیت‌ها استفاده می‌شوند.


```
value
    : STRING | INT | FLOAT;

entityName
    : ID;

STRING: '"' (ESC | .)*? '"';
FLOAT: DIGIT+ '.' DIGIT* | DIGIT* '.' DIGIT+;
INT: DIGIT+;
ID: LETTER (LETTER | DIGIT | '_' )*
```

- value: یک مقدار می‌تواند از نوع رشته (STRING)، عدد صحیح (INT)، یا عدد اعشاری (FLOAT) باشد.
- STRING: نشان‌دهنده مقادیر رشته‌ای است که با علامت نقل قول (") احاطه شده‌اند. مقادیر می‌توانند شامل کاراکترهای خاص باشند که از طریق ESC مشخص می‌شوند.
- FLOAT: اعداد اعشاری هستند که شامل یک یا چند رقم قبل و بعد از نقطه اعشاری می‌شوند.
- INT: اعداد صحیح که فقط شامل ارقام هستند.

۱۱. کاراکترهای پایه: DIGIT, LETTER و ESC

```
fragment DIGIT: [0-9];
fragment LETTER: [a-zA-Z];
fragment ESC: '\\\"' | '\\\\';
```

- DIGIT: هر عدد بین ۰ تا ۹.
- LETTER: هر حرف الفبای انگلیسی (کوچک یا بزرگ).
- ESC: کاراکترهای خاصی که باید در رشته‌ها فرار داده شوند:

- \: نقل قول فرار.

- \\: بک اسلش فرار.

۱۲. COMMENT

```
COMMENT: ('//'.*?'\\n' | '/*'.*?'*/');
```

- کامنت تک خطی: با // شروع شده و تا انتهای خط ادامه می یابد.

- کامنت چندخطی: با / شروع و با / پایان می یابد.

۱۳. فضای خالی و خطوط جدید: WS و NEWLINE

```
WS: [ \t\r\n]+ -> skip;  
NEWLINE: ('\\n' | '\\r\\n' | '\\r') -> skip;
```

- WS: نشان دهنده فضاهای خالی، تبها، و خطوط جدید است. این کاراکترها در تجزیه نادیده گرفته می شوند (با استفاده از -> skip).

- NEWLINE: نشان دهنده پایان خط یا خطوط جدید است که در تجزیه نادیده گرفته می شوند.

توضیح یک مثال مربوط به ورودی و خروجی پروژه

ورودی شامل مجموعه ای از دستورات DSL است که وظایف مختلفی را برای مدیریت فروشگاه آنلاین انجام می دهند. در ادامه یک مثال را به صورت کامل بررسی میکنیم.

نمونه ورودی

تعریف موجودیت ها: (define)

```
define {  
  entity: Order,  
  attributes {  
    id: String,  
    customer_name: String,  
    total_amount: Float,  
    order_date: String,  
    status: String  
  }  
}
```

○ موجودیت Order با ویژگی های مشخص

تعریف شده اند.

○ نمونه دستور:

ایجاد موجودیت‌ها: (create)

- یک سفارش (Order) و یک مشتری (Customer) با مقادیر مشخص ایجاد شده‌اند.

```
create {
  entity: Order,
  attributes {
    id: "01001",
    customer_name: "John Doe",
    total_amount: 250.0,
    order_date: "2025-01-25",
    status: "Pending"
  }
}
```

- نمونه دستور:

به‌روزرسانی: (update)

- وضعیت سفارش با id مشخص به‌روزرسانی شده است.

```
update {
  entity: Order,
  attributes {
    id: "01001",
    status: "Shipped",
    order_date: "1999,-12-23"
  }
}
```

- نمونه دستور:

حذف موجودیت‌ها: (delete)

- مشتری با id مشخص حذف شده است.

```
delete {
  entity: Customer,
  attributes {
    id: "C1001"
  }
}
```

- نمونه دستور:

مشاهده لیست (list):

```
list {  
    entity: Order  
}
```

○ لیست سفارش‌ها درخواست شده است.

○ نمونه دستور:

تولید کد برای ورودی مثال زده شده

خروجی تولید شده شامل کد Python است که با استفاده از فریمورک Flask و SQLAlchemy وظایف تعریف‌شده در DSL را پیاده‌سازی می‌کند. در ادامه بخش‌های مختلف این کد توضیح داده شده‌اند:

۱. وارد کردن کتابخانه‌های مورد نیاز

```
import json  
import logging  
from flask import Flask, request, jsonify  
from flask_sqlalchemy import SQLAlchemy  
from flask_cors import CORS # Import CORS  
from random import randint
```

- Flask: برای ساخت API و اجرای سرور.

- SQLAlchemy: برای تعامل با پایگاه داده.

- CORS: برای پشتیبانی از درخواست‌های Cross-Origin.

- json و logging: برای مدیریت داده‌های JSON و ثبت لاگ.

۲. تنظیمات اولیه Flask و SQLAlchemy

```
app = Flask(__name__)
CORS(app)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///store.db'
db = SQLAlchemy(app)
```

- یک برنامه Flask ایجاد شده است.

- CORS برای مدیریت درخواست‌های Cross-Origin فعال شده است.

- مسیر پایگاه داده SQLite تنظیم شده و SQLAlchemy برای مدیریت مدل‌ها استفاده می‌شود.

۳. تعریف مدل‌ها

```
class Order(db.Model):
    id = db.Column(db.String, primary_key=True)
    customer_name = db.Column(db.String)
    total_amount = db.Column(db.Float)
    order_date = db.Column(db.String)
    status = db.Column(db.String)

class Customer(db.Model): 4 usages
    id = db.Column(db.String, primary_key=True)
    full_name = db.Column(db.String)
    email = db.Column(db.String)
    phone_number = db.Column(db.String)
```

- Order: مدلی برای سفارش‌ها با ویژگی‌هایی مانند id, customer_name, total_amount, order_date و status.

- Customer: مدلی برای مشتریان با ویژگی‌هایی مانند id, full_name, email و phone_number.

۴. ایجاد پایگاه داده

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

- با ایجاد کانتکست برنامه Flask، تمام جداول پایگاه داده بر اساس مدل‌ها ایجاد می‌شوند.

۵. پیاده‌سازی عملیات (CRUD)

ایجاد (Create)

- سفارش‌ها و مشتریان جدید به پایگاه داده اضافه می‌شوند.

```
try:  
    order = Order(  
        id = '01001',  
        customer_name = 'John Doe',  
        total_amount = '250.0',  
        order_date = '2025-01-25',  
        status = 'Pending'  
    )  
    db.session.add(order)  
    db.session.commit()  
except:  
    pass
```

به‌روزرسانی (Update)

- ویژگی‌های یک سفارش موجود به‌روزرسانی می‌شوند.

```
try:  
    order = Order.query.get('01001')  
    if order:  
        order.status = 'Shipped'  
        order.order_date = '2025-01-25'  
        db.session.commit()
```

حذف (Delete)

- مشتری با id مشخص از پایگاه داده حذف می‌شود.

```
try:
    customer = Customer.query.get('C1001')
    if customer:
        db.session.delete(customer)
        db.session.commit()
```

مشاهده لیست (List)

- لیست تمام سفارش‌ها از پایگاه داده بازیابی می‌شود.

```
def list_orders():
    orders = Order.query.all()
    order_list = [{
        "id": order.id,
        "customer_name": order.customer_name,
        "total_amount": order.total_amount,
        "order_date": order.order_date,
        "status": order.status
    } for order in orders]
    return jsonify(order_list), 200
```

۶. تعریف مسیرهای API

- برای هر عملیات یک مسیر API مشخص شده است:

ایجاد سفارش:

```
@app.route('/createOrder', methods=['POST'])
def create_order():
```

ایجاد مشتری:

```
@app.route('/createCustomer', methods=['POST'])
def create_customer():
```

به‌روزرسانی سفارش:

```
@app.route('/updateOrder', methods=['POST'])
def update_order():
```

حذف مشتری:

```
@app.route('/deleteCustomer', methods=['POST'])
def delete_customer():
```

مشاهده لیست سفارش‌ها:

```
@app.route('/listOrders', methods=['GET'])
def list_orders():
```


منابع و ابزارهای استفاده شده

۱. ANTLR v4:

- برای طراحی و تولید Lexer و Parser.
- لینک: (https://github.com/antlr/antlr4)[ANTLR GitHub]

۲. Python:

- زبان اصلی برای پیاده‌سازی کامپایلر و عملیات.
- کتابخانه‌های مورد استفاده:
- Flask: برای ایجاد API و مدیریت سرور.
- SQLAlchemy: برای مدیریت پایگاه داده.
- Flask-CORS: برای مدیریت درخواست‌های Cross-Origin.

۳. SQLite:

- پایگاه داده مورد استفاده برای ذخیره‌سازی اطلاعات.

۴. ابزارهای توسعه:

- Visual Studio Code یا PyCharm: برای توسعه و ویرایش کد.
- Postman: برای تست و بررسی API‌های تولید شده.