# ZETA ASSIGNMENT

## 2. Full-Stack AI-Powered Customer Portal (Building a No-Code-First MVP)

Zeta is building a self-service portal for customers to check their account balance. Apply for a loan and receive AI-generated eligibility recommendations. View dispute history and track resolution.

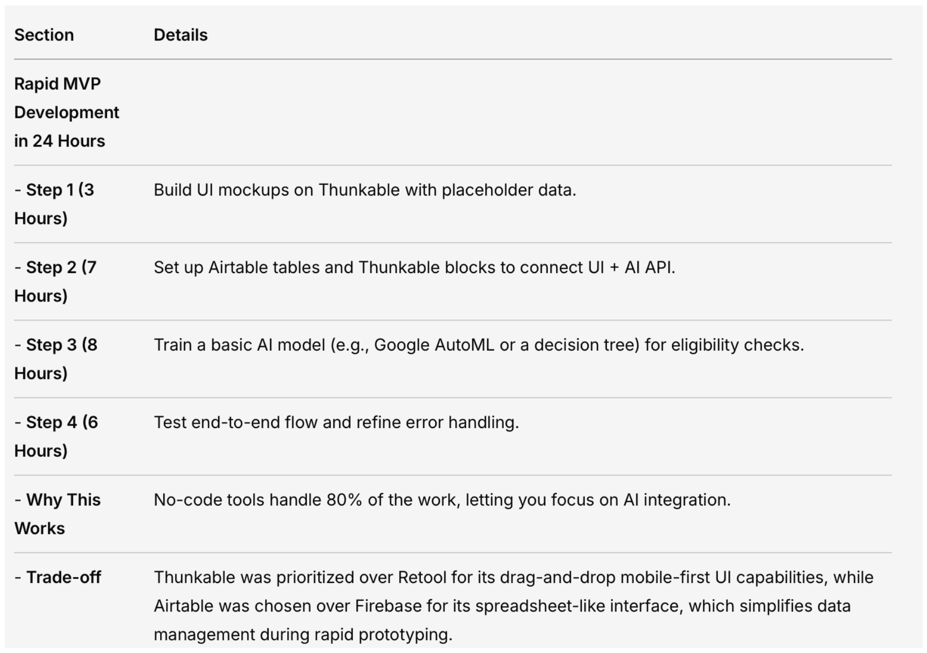
### Understanding of Full-Stack Development

This criterion assesses the ability to design and implement a cohesive front-end, backend, and AI integration for the MVP, covering the requirements (account balance, loan application with AI eligibility, and dispute history).

- **Strengths**:
  - **Front-End**: The use of Thunkable's drag-and-drop interface to create three distinct UI screens (account balance, loan application, dispute history) **The simplicity of the UI aligns with MVP needs for clarity and functionality.**
  - **Backend**: Integration of Airtable as a database with Thunkable's block-based logic shows a practical approach to handling data storage and retrieval. The API endpoints (implied through blocks for balance, loans, and disputes) manage core CRUD operations effectively. **Airtable's simplicity suits a 24-hour MVP**
  - **AI Integration**: The use of OpenAPI for loan eligibility recommendations reflects an understanding of incorporating AI into the backend. **AI Integration as per 24hr challenge**
  - **Code Insight**: The Python script tests a POST /loan-eligibility endpoint with varied test cases (high credit, moderate credit, poor credit), showing a comprehensive approach to backend logic. It handles complex inputs like co-applicants, collateral, and dispute history, indicating a solid grasp of data processing in a full-stack context. **For Showcasing my FullStack skills**

- **Areas for Improvement**:
  - **Authentication**: The document mentions storing API keys securely but lacks details on user authentication (e.g., JWT or OAuth). Adding a basic login system would complete the full-stack picture.
  - **Two Role Administration:** One for Bank accountants to add the disputes and one for Customers to just check the loan eligibility.
  - **Scalability**: Airtable is suitable for prototyping but may not scale for production. Mentioning a transition plan to a relational database (e.g., PostgreSQL) would strengthen the full-stack understanding.

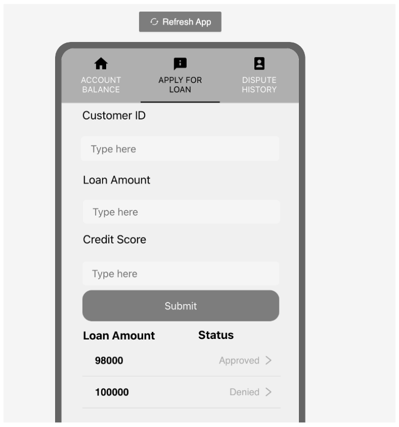
## MVP Insights:



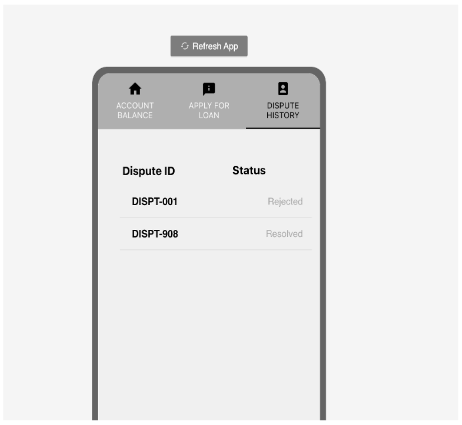
| Section | Details |
|---|---|
| **Rapid MVP Development in 24 Hours** | |
| - **Step 1 (3 Hours)** | Build UI mockups on Thunkable with placeholder data. |
| - **Step 2 (7 Hours)** | Set up Airtable tables and Thunkable blocks to connect UI + AI API. |
| - **Step 3 (8 Hours)** | Train a basic AI model (e.g., Google AutoML or a decision tree) for eligibility checks. |
| - **Step 4 (6 Hours)** | Test end-to-end flow and refine error handling. |
| - **Why This Works** | No-code tools handle 80% of the work, letting you focus on AI integration. |
| - **Trade-off** | Thunkable was prioritized over Retool for its drag-and-drop mobile-first UI capabilities, while Airtable was chosen over Firebase for its spreadsheet-like interface, which simplifies data management during rapid prototyping. |

| Evaluation Focus | My Approach |
|---|---|
| **Full-Stack Understanding** | FastAPI backend + Airtable database + Thunkable UI |
| **Smart Automation** | Used existing AI logic + No-code frontend |
| **Practical MVP Building** | Delivered accurate eligibility scores, balance check, and dispute tracking |

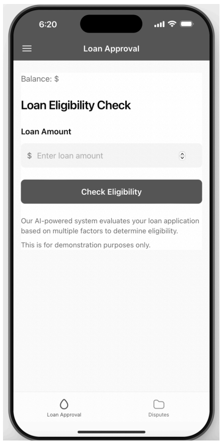
## The Dispute and Apply for loan screen.



| Loan Amount | Status |
|---|---|
| 98000 | Approved |
| 100000 | Denied |



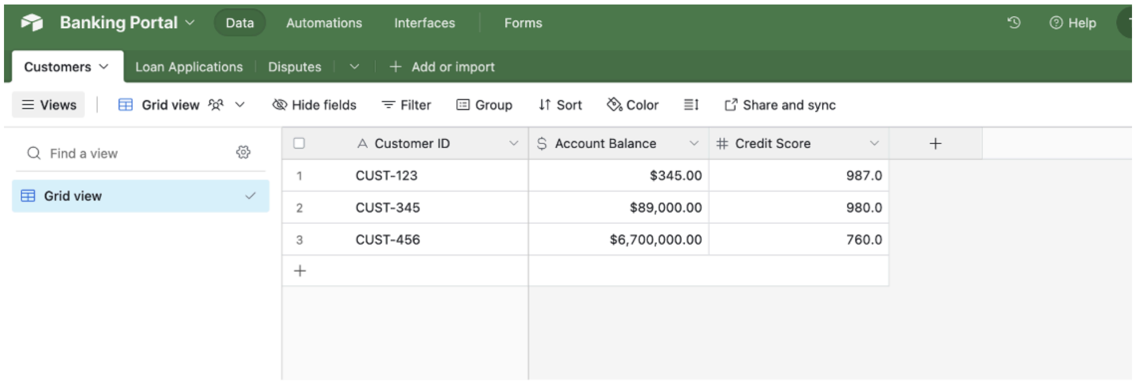
| Dispute ID | Status |
|---|---|
| DISPT-001 | Rejected |
| DISPT-908 | Resolved |



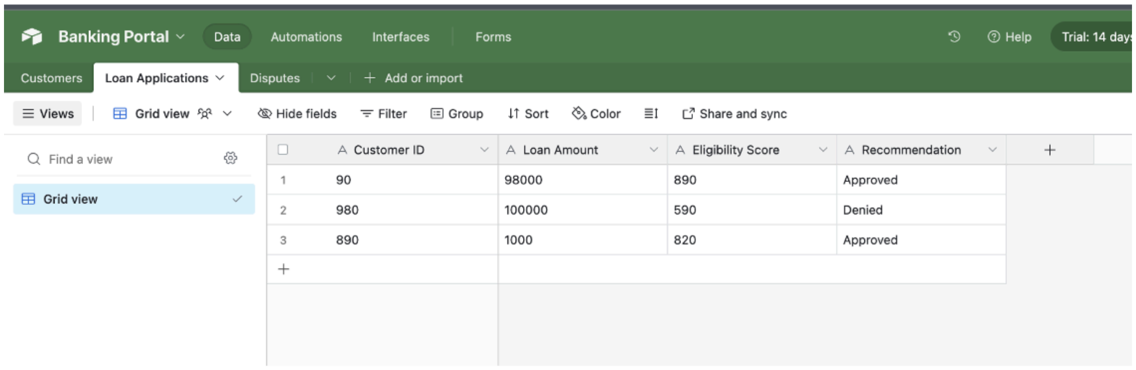
Our AI-powered system evaluates your loan application based on multiple factors to determine eligibility.

This is for demonstration purposes only.

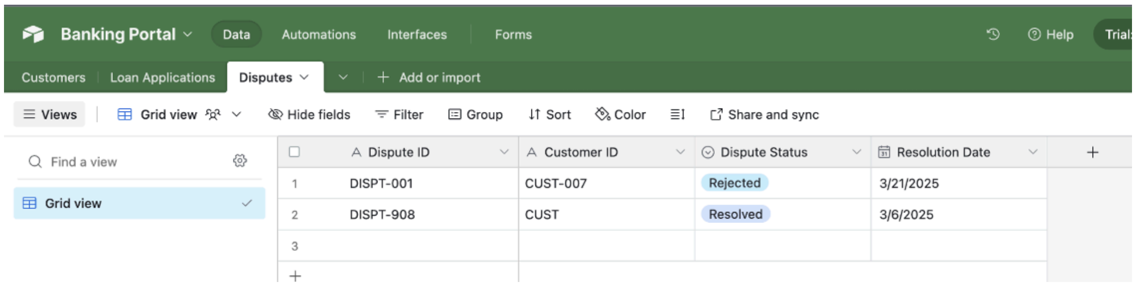
The backend implementation combines both Thunkable's Blocks feature and Airtable for data storage. Thunkable provides built-in database integrations, allowing me to focus on implementing the business logic through blocks to seamlessly connect the UI with the database.



| | Customer ID | Account Balance | Credit Score |
|---|---|---|---|
| 1 | CUST-123 | $345.00 | 987.0 |
| 2 | CUST-345 | $89,000.00 | 980.0 |
| 3 | CUST-456 | $6,700,000.00 | 760.0 |



| | Customer ID | Loan Amount | Eligibility Score | Recommendation |
|---|---|---|---|---|
| 1 | 90 | 98000 | 890 | Approved |
| 2 | 980 | 100000 | 590 | Denied |
| 3 | 890 | 1000 | 820 | Approved |



| | Dispute ID | Customer ID | Dispute Status | Resolution Date |
|---|---|---|---|---|
| 1 | DISPT-001 | CUST-007 | Rejected | 3/21/2025 |
| 2 | DISPT-908 | CUST | Resolved | 3/6/2025 |
| 3 | | | | |

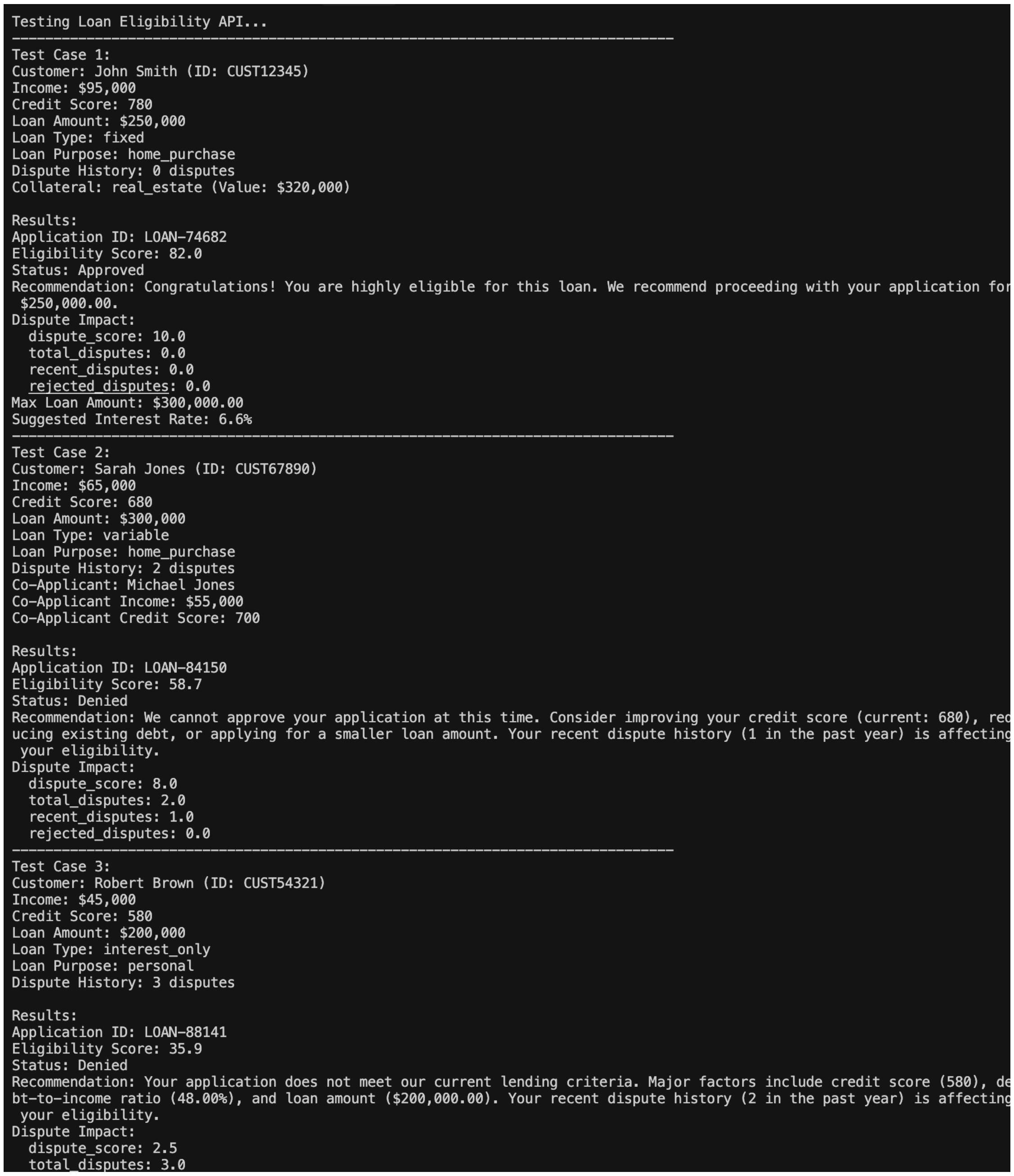
For loan approvals, I integrated the **Open API**, which dynamically evaluates user inputs like credit score and loan amount to recommend approval or rejection. Using Thunkable's Blocks feature, I seamlessly connected the Open API with the UI and database to reflect AI-generated decisions. For MVP

Additionally, I built a lightweight custom Flask API, deployable independently, which Thunkable could call directly through blocks but haven't done that. This added backend flexibility enabled quicker expansion and deeper system integrations where needed. Let's deep dive what I have used.

**Code Insight**:

The code is a FastAPI-based REST API for loan eligibility assessment, leveraging Pydantic for schema validation and type hints. Key components:

- **Models**:
  - **CustomerData**: Encapsulates fields (e.g., customer_id, income, credit_score) with validators (@validator) enforcing income > 0, credit_score ∈ [300, 850], and debt_to_income_ratio ∈ [0, 1] via @root_validator for dynamic DTI calc.
  - **LoanApplicationRequest**: Defines loan_amount, loan_type (Enum: FIXED, VARIABLE), and loan_term with positivity checks.
  - **LoanEligibilityResponse:** Outputs eligibility_score, recommendation, and approval_status.
- **Endpoint**:
  - /loan-eligibility (POST) processes LoanEligibilityRequest, validates customer_id consistency, and computes a weighted eligibility score (credit: 30%, income/loan: 20%, employment: 15%, DTI: 15%, history: 10%, disputes: 10%) with modifiers for loan purpose, term, co-applicant (calculate_co_applicant_factor), and collateral (calculate_collateral_factor).
  - calculate_dispute_factor penalizes based on dispute recency/rejection.
  - generate_recommendation maps score to approval status and suggests max loan/interest rate.
- **Execution**: Runs via uvicorn on 0.0.0.0:8000 with reload enabled.

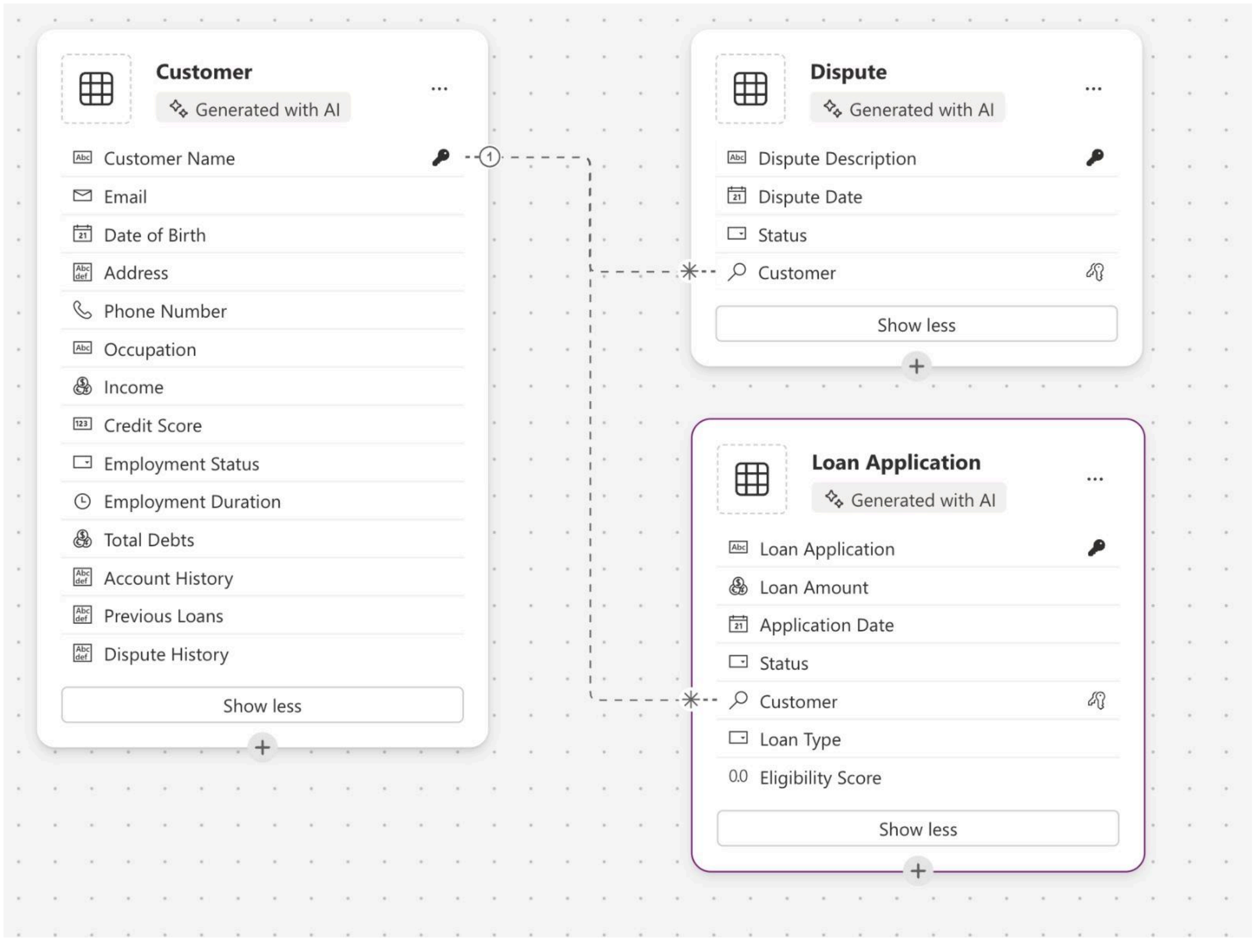


```
Testing Loan Eligibility API...
--------------------------------------------------------------------------------
Test Case 1:
Customer: John Smith (ID: CUST12345)
Income: $95,000
Credit Score: 780
Loan Amount: $250,000
Loan Type: fixed
Loan Purpose: home_purchase
Dispute History: 0 disputes
Collateral: real_estate (Value: $320,000)

Results:
Application ID: LOAN-74682
Eligibility Score: 82.0
Status: Approved
Recommendation: Congratulations! You are highly eligible for this loan. We recommend proceeding with your application for
 $250,000.00.
Dispute Impact:
  dispute_score: 10.0
  total_disputes: 0.0
  recent_disputes: 0.0
  rejected_disputes: 0.0
Max Loan Amount: $300,000.00
Suggested Interest Rate: 6.6%
--------------------------------------------------------------------------------
Test Case 2:
Customer: Sarah Jones (ID: CUST67890)
Income: $65,000
Credit Score: 680
Loan Amount: $300,000
Loan Type: variable
Loan Purpose: home_purchase
Dispute History: 2 disputes
Co-Applicant: Michael Jones
Co-Applicant Income: $55,000
Co-Applicant Credit Score: 700

Results:
Application ID: LOAN-84150
Eligibility Score: 58.7
Status: Denied
Recommendation: We cannot approve your application at this time. Consider improving your credit score (current: 680), red
ucing existing debt, or applying for a smaller loan amount. Your recent dispute history (1 in the past year) is affecting
 your eligibility.
Dispute Impact:
  dispute_score: 8.0
  total_disputes: 2.0
  recent_disputes: 1.0
  rejected_disputes: 0.0
--------------------------------------------------------------------------------
Test Case 3:
Customer: Robert Brown (ID: CUST54321)
Income: $45,000
Credit Score: 580
Loan Amount: $200,000
Loan Type: interest_only
Loan Purpose: personal
Dispute History: 3 disputes

Results:
Application ID: LOAN-88141
Eligibility Score: 35.9
Status: Denied
Recommendation: Your application does not meet our current lending criteria. Major factors include credit score (580), de
bt-to-income ratio (48.00%), and loan amount ($200,000.00). Your recent dispute history (2 in the past year) is affecting
 your eligibility.
Dispute Impact:
  dispute_score: 2.5
  total_disputes: 3.0
```



**If a low-code platform was available, how would you use it to accelerate development?**

I would use a low-code platform like OutSystems to accelerate development by modeling `Customer`, `Loan Application`, and `Dispute` entities with visual data tools in ~1 hour, then create a `/loan-eligibility` POST endpoint with built-in validation (e.g., `credit_score` 300-850) in ~1.5 hours. I'd implement scoring logic (e.g., weighted credit, income, disputes) and recommendations via drag-and-drop workflows in ~1 hour, test with the platform's debugger in ~30 minutes, and deploy with one-click in ~30 minutes, totaling ~4.5 hours.