Flag Learning App - Design Document v1.2

Last Updated: October 22, 2025
Status: Pre-Development / Planning Complete
Developer: Solo Project
Target Launch: 4-5 weeks (MVP)

Executive Summary

A geography learning application focused on flag recognition through daily challenges. Users receive one flag per day to identify, with their progress tracked across iOS app, web platform, and iOS widget. The app emphasizes gradual learning through spaced repetition and gamification elements.

Core Value Proposition:

- Daily micro-learning habit (< 2 minutes per day)
- Progressive difficulty with no pressure (3 attempts per flag)
- Rich educational content beyond simple recognition
- Cross-platform consistency (web, iOS, widget)

Development Philosophy:

- MVP-first approach with architecture designed for extensibility
- Learn Django and React ecosystem while building production-quality code
- No feature creep in MVP, but future enhancements designed into architecture from day one

---

Technical Stack

Backend

- **Framework:** Django 5.0+ with Django REST Framework

- **Language:** Python 3.11+
- **Database:**
  - Development: SQLite
  - Production: PostgreSQL
- **Authentication:** OAuth 2.0 (Google Sign-In initially)
- **API Style:** RESTful with JSON responses
- **Hosting:** TBD at deployment (Railway, Render, or AWS)

Frontend - Web

- **Framework:** React 18+
- **Language:** JavaScript (ES6+)
- **Styling:** Material-UI (MUI)
- **State Management:** Zustand
- **HTTP Client:** Axios
- **Routing:** React Router

Frontend - Mobile

- **Framework:** React Native with Expo
- **Styling:** React Native Paper (MUI design language equivalent)
- **State Management:** Zustand (100% shared with web)
- **Navigation:** React Navigation

Code Reusability Note: UI components require rewriting (~30% of code) due to platform differences, but all business logic, state management, API services, and utilities are 100% reusable between web and mobile. MUI and React Native Paper share similar design language and component patterns, ensuring visual consistency.

Frontend - Widget

- **Framework:** Native iOS WidgetKit (Swift)
- **Note:** Separate codebase from React Native app

External Services

- **Country Data:** REST Countries API (https://restcountries.com)
- **Maps:** Leaflet (web) / React Native Maps (mobile)
- **OAuth Provider:** Google Identity Services

Architecture Overview

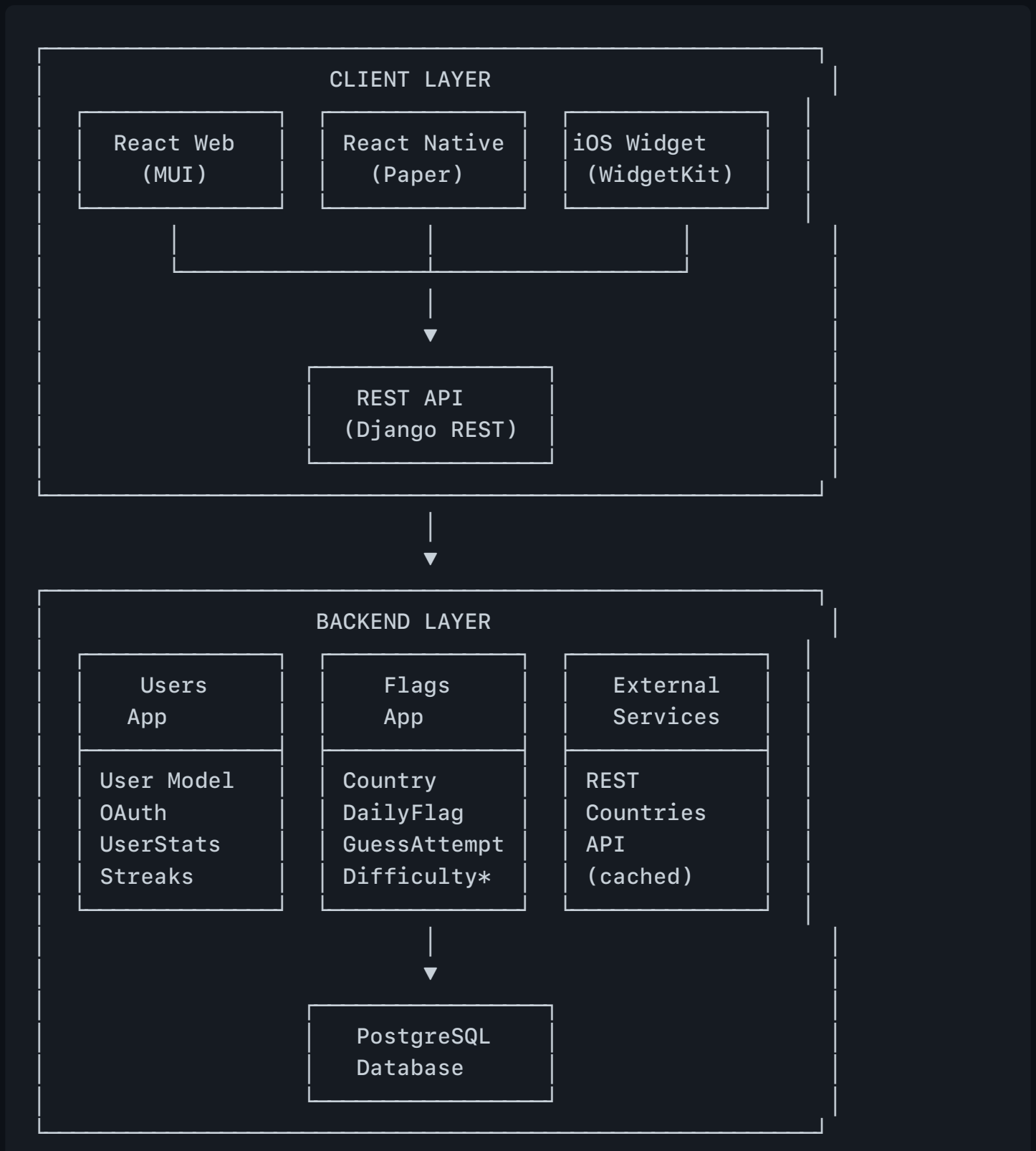
Design Principles for Extensibility

The architecture is designed with these future capabilities in mind:

1. **Pluggable Difficulty System**
   - Current: Single random pool
   - Future: Tiered difficulty with rotation
   - Design: Country model includes `difficulty_tier` field (nullable in MVP)
2. **Quiz Mode Foundation**
   - Current: Daily single-flag challenge
   - Future: Multiple quiz modes (speed rounds, regional focus, etc.)
   - Design: Separation of `DailyChallenge` from `GuessAttempt` models
3. **Multiple OAuth Providers**
   - Current: Google only
   - Future: Apple, GitHub, email/password
   - Design: Using `django-allauth` which supports 50+ providers
4. **Enhanced Gamification**
   - Current: Basic streaks and stats
   - Future: Achievements, leaderboards, social features
   - Design: Extensible stats system with JSON field for future metrics
5. **Content Expansion**
   - Current: Country data from API

- Future: Custom trivia, historical facts, regional groupings
- Design: Flexible CountryMetadata model for custom fields

System Architecture



```
* Difficulty system implemented but not active in MVP
```

Data Models

Core Models (MVP)

User Model (extends Django AbstractUser)

```
class User(AbstractUser):
    # OAuth fields handled by django-allauth
    email = EmailField(unique=True)
    created_at = DateTimeField(auto_now_add=True)
    last_login = DateTimeField(auto_now=True)
```

UserStats Model

```
class UserStats(Model):
    user = OneToOneField(User)

    # Daily challenge stats
    total_correct = IntegerField(default=0)
    current_streak = IntegerField(default=0)
    longest_streak = IntegerField(default=0)
    last_guess_date = DateField(null=True)
    incorrect_countries = JSONField(default=list)  # List of country
codes

    # FUTURE: Per-category quiz stats (Phase 3)
    # Structure: {
    #   "flag": {
    #     "correct": 50,
    #     "total": 100,
    #     "accuracy": 0.50,
    #     "by_format": {
```

```
    #       "text_input": {"correct": 30, "total": 50, "accuracy":
0.60},
    #       "multiple_choice": {"correct": 20, "total": 50, "accuracy":
0.40}
    #     }
    #   },
    #   "capital": {...},
    #   ...
    # }
    category_stats = JSONField(default=dict)

    # FUTURE: Per-format stats (across all categories)
    # Structure: {
    #   "text_input": {"correct": 80, "total": 150, "accuracy": 0.53},
    #   "multiple_choice": {"correct": 95, "total": 120, "accuracy":
0.79},
    #   "true_false": {"correct": 40, "total": 50, "accuracy": 0.80}
    # }
    format_stats = JSONField(default=dict)

    # FUTURE: Extended stats (Phase 4+)
    extra_stats = JSONField(default=dict)  # For future metrics without
migrations

    class Meta:
        verbose_name_plural = "User Stats"

    def update_daily_streak(self, is_correct, guess_date):
        """Update streak based on today's guess"""
        if is_correct:
            if self.last_guess_date == guess_date - timedelta(days=1):
                self.current_streak += 1
            else:
                self.current_streak = 1
            self.longest_streak = max(self.longest_streak,
self.current_streak)
            self.total_correct += 1
        else:
            self.current_streak = 0
        self.last_guess_date = guess_date
        self.save()
```

```python
    def update_category_stat(self, category, format_type, is_correct):
        """
        Update per-category and per-format statistics (Phase 3).
        Tracks both overall category performance and breakdown by
format.
        """
        # Initialize category if not exists
        if category not in self.category_stats:
            self.category_stats[category] = {
                "correct": 0,
                "total": 0,
                "accuracy": 0.0,
                "by_format": {}
            }

        # Update category totals
        cat_stats = self.category_stats[category]
        cat_stats["total"] += 1
        if is_correct:
            cat_stats["correct"] += 1
        cat_stats["accuracy"] = cat_stats["correct"] /
cat_stats["total"]

        # Update category breakdown by format
        if format_type not in cat_stats["by_format"]:
            cat_stats["by_format"][format_type] = {"correct": 0,
"total": 0, "accuracy": 0.0}

        fmt_stats = cat_stats["by_format"][format_type]
        fmt_stats["total"] += 1
        if is_correct:
            fmt_stats["correct"] += 1
        fmt_stats["accuracy"] = fmt_stats["correct"] /
fmt_stats["total"]

        # Update global format stats
        if format_type not in self.format_stats:
            self.format_stats[format_type] = {"correct": 0, "total": 0,
"accuracy": 0.0}
```

```
        global_fmt = self.format_stats[format_type]
        global_fmt["total"] += 1
        if is_correct:
            global_fmt["correct"] += 1
        global_fmt["accuracy"] = global_fmt["correct"] /
global_fmt["total"]

        self.save()

    def get_weakest_categories(self, limit=5):
        """Returns categories where user performs worst"""
        if not self.category_stats:
            return []

        sorted_categories = sorted(
            self.category_stats.items(),
            key=lambda x: x[1]["accuracy"]
        )
        return [(cat, stats["accuracy"]) for cat, stats in
sorted_categories[:limit]]

    def get_strongest_formats(self, limit=3):
        """Returns question formats user is best at"""
        if not self.format_stats:
            return []

        sorted_formats = sorted(
            self.format_stats.items(),
            key=lambda x: x[1]["accuracy"],
            reverse=True
        )
        return [(fmt, stats["accuracy"]) for fmt, stats in
sorted_formats[:limit]]
```

Analytics Benefits:

- Track performance by category (flags, capitals, etc.)
- Track performance by format (text input, multiple choice, T/F)
- Track combination (e.g., "I'm bad at flag text input but good at flag multiple choice")

- Identify weak areas for targeted practice
- Recommend formats that user excels at

Question Generation System

Adding New Question Categories

Adding a new category is a **three-step process** with no database migrations required:

Step 1: Add to Enum

```
# In models.py
class QuestionCategory(models.TextChoices):
    FLAG = 'flag', 'Flag Recognition'
    CAPITAL = 'capital', 'Capital City'
    # ... existing categories ...

    # NEW CATEGORY - just add here!
    BORDER_COUNTRIES = 'border_countries', 'Bordering Countries'
```

Step 2: Add Generator Function

```
# In question_generators.py
def generate_border_countries_question(country, format_type):
    """Generate a question about bordering countries"""

    if format_type == QuestionFormat.TEXT_INPUT:
        return Question(
            category=QuestionCategory.BORDER_COUNTRIES,
            format=QuestionFormat.TEXT_INPUT,
            country=country,
            question_text=f"Name a country that borders
{country.name}",
            correct_answer={
                "answer": country.border_countries[0],
                "alternates": country.border_countries[1:]
```

```
            }
        )

    elif format_type == QuestionFormat.MULTIPLE_CHOICE:
        correct = random.choice(country.border_countries)
        wrong_options = get_random_countries(3, exclude=[country,
correct])
        options = [correct] + wrong_options
        random.shuffle(options)

        return Question(
            category=QuestionCategory.BORDER_COUNTRIES,
            format=QuestionFormat.MULTIPLE_CHOICE,
            country=country,
            question_text=f"Which country borders {country.name}?",
            correct_answer={
                "correct": correct.name,
                "options": [c.name for c in options]
            }
        )
```

Step 3: Register Generator

```
# In question_generators.py
CATEGORY_GENERATORS = {
    QuestionCategory.FLAG: generate_flag_question,
    QuestionCategory.CAPITAL: generate_capital_question,
    QuestionCategory.BORDER_COUNTRIES:
generate_border_countries_question,  # NEW!
}
```

Done! The new category is now available throughout the app.

Adding New Question Formats

Adding a new format is also a **three-step process**:

Step 1: Add to Enum

```python
# In models.py
class QuestionFormat(models.TextChoices):
    TEXT_INPUT = 'text_input', 'Text Input'
    MULTIPLE_CHOICE = 'multiple_choice', 'Multiple Choice'
    TRUE_FALSE = 'true_false', 'True/False'
    MAP_LOCATION = 'map_location', 'Locate on Map'  # NEW!
```

Step 2: Add Validation Logic

```python
# In Question.validate_answer() method
elif self.format == QuestionFormat.MAP_LOCATION:
    user_lat = user_answer_data.get('latitude')
    user_lng = user_answer_data.get('longitude')
    correct_lat = self.correct_answer['latitude']
    correct_lng = self.correct_answer['longitude']
    tolerance = self.correct_answer.get('tolerance_km', 100)

    distance = calculate_distance(user_lat, user_lng, correct_lat,
correct_lng)
    is_correct = distance <= tolerance
    return is_correct, f"Location: {correct_lat}, {correct_lng}"
```

Step 3: Update Generator Functions

```python
# Each category generator checks format and creates appropriate
question
def generate_capital_question(country, format_type):
    if format_type == QuestionFormat.MAP_LOCATION:  # NEW!
        capital_coords = get_city_coordinates(country.capital)
        return Question(
            category=QuestionCategory.CAPITAL,
            format=QuestionFormat.MAP_LOCATION,
            country=country,
            question_text=f"Locate {country.capital} on the map",
            correct_answer={
                "latitude": capital_coords.lat,
                "longitude": capital_coords.lng,
```

```
                "tolerance_km": 50
            }
        )
```

Done! The new format is now available for all compatible categories.

Question Generation Examples

Text Input - Flag Recognition

```
{
    "category": "flag",
    "format": "text_input",
    "question_text": "Which country does this flag belong to? 🇫🇷",
    "correct_answer": {
        "answer": "France",
        "alternates": ["france", "French Republic"]
    }
}
```

Multiple Choice - Capital

```
{
    "category": "capital",
    "format": "multiple_choice",
    "question_text": "What is the capital of Japan?",
    "correct_answer": {
        "correct": "Tokyo",
        "options": ["Tokyo", "Osaka", "Kyoto", "Seoul"]
    }
}
```

True/False - Language

```
{
    "category": "language",
    "format": "true_false",
    "question_text": "English is the primary language of the United
States",
    "correct_answer": {
        "answer": true
    }
}
```

Future: Map Location - Capital

```
{
    "category": "capital",
    "format": "map_location",
    "question_text": "Locate Paris on the map",
    "correct_answer": {
        "latitude": 48.8566,
        "longitude": 2.3522,
        "tolerance_km": 50
    }
}
```

Quiz Configuration System (Phase 3)

Users can fully customize their quiz experience with format selection:

Frontend Quiz Creation

```
const quizConfig = {
  question_count: 10,
  selected_categories: ['flag', 'capital', 'population'],
  selected_formats: ['multiple_choice', 'true_false'],  // or null for
any
  difficulty_filter: 'medium',
  region_filter: null
};
```

Format Selection Options:

- **"Any format"** (null) - System chooses best format per question
- **"Multiple choice only"** - All questions use multiple choice
- **"Text input + True/False"** - Mix of two formats
- **"All formats"** - Maximum variety

Country Model

```
class Country(Model):
    code = CharField(max_length=3, unique=True)  # ISO 3166-1 alpha-3
    name = CharField(max_length=100)
    flag_emoji = CharField(max_length=10)

    # Core data (from REST Countries API - explicitly stored fields)
    population = BigIntegerField()
    capital = CharField(max_length=100)
    largest_city = CharField(max_length=100)
    languages = JSONField()  # ["English", "Spanish"]
    area_km2 = FloatField()
    currencies = JSONField()

    # Extended data
    gdp_nominal = BigIntegerField(null=True)
    gdp_ppp_per_capita = IntegerField(null=True)
    median_age = FloatField(null=True)
    life_expectancy = FloatField(null=True)
```

```
    school_expectancy = FloatField(null=True)
    religions = JSONField(null=True)
    arable_land_percent = FloatField(null=True)
    fertility_rate = FloatField(null=True)
    highest_point = CharField(max_length=100, null=True)

    # Map coordinates
    latitude = FloatField()
    longitude = FloatField()

    # FUTURE: Difficulty tier (not used in MVP)
    difficulty_tier = CharField(
        max_length=10,
        choices=[('easy', 'Easy'), ('medium', 'Medium'), ('hard',
'Hard')],
        null=True,
        blank=True
    )
    popularity_score = IntegerField(default=0)  # For future tier
assignment

    # API caching strategy
    api_cache_date = DateTimeField()  # When we last fetched from REST
Countries
    raw_api_response = JSONField(null=True, blank=True)  # Full JSON
from API for flexibility

    class Meta:
        indexes = [
            Index(fields=['difficulty_tier']),  # For future tier
queries
            Index(fields=['name']),  # For encyclopedia search
        ]

    def __str__(self):
        return f"{self.flag_emoji} {self.name}"
```

Caching Strategy: We store explicitly-defined fields in the schema for fast queries and type safety. The `raw_api_response` field stores the complete JSON from REST Countries API as a fallback. This allows us to access any field we didn't anticipate needing without additional API calls or migrations.

DailyChallenge Model

```
class DailyChallenge(Model):
    """
    Represents the daily flag challenge.
    Links to a Question which contains the actual question data.
    """
    date = DateField(unique=True)
    country = ForeignKey(Country)

    # FUTURE: Difficulty tracking (not used in MVP)
    difficulty_tier = CharField(max_length=10, null=True)

    # Algorithm tracking
    selection_algorithm_version = CharField(max_length=20,
default='v1_random')

    created_at = DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-date']

    def __str__(self):
        return f"{self.date} - {self.country.flag_emoji}
{self.country.name}"

    def get_or_create_question(self):
        """
        Get or create the Question for this daily challenge.
        MVP: Always creates a flag recognition text input question.
        """
        question, created = Question.objects.get_or_create(
            daily_challenge=self,
            defaults={
                'category': QuestionCategory.FLAG,
```

```
                'format': QuestionFormat.TEXT_INPUT,
                'country': self.country,
                'question_text': f"Which country does this flag belong
to? {self.country.flag_emoji}",
                'correct_answer': {
                    'answer': self.country.name,
                    'alternates': [
                        # Can add common alternate names
                        # e.g., "USA" for "United States"
                    ]
                }
            }
        )
        return question
```

Note: In MVP, daily challenges use simple text input flag questions. In Phase 3, you could vary the daily challenge format (some days multiple choice, some days T/F, etc.).

Question Model

```
# Extensible category choices - add new categories here
class QuestionCategory(models.TextChoices):
    FLAG = 'flag', 'Flag Recognition'
    CAPITAL = 'capital', 'Capital City'
    POPULATION = 'population', 'Population'
    CURRENCY = 'currency', 'Currency'
    LANGUAGE = 'language', 'Language(s)'
    AREA = 'area', 'Area Size'
    LARGEST_CITY = 'largest_city', 'Largest City'
    CONTINENT = 'continent', 'Continent/Region'
    HIGHEST_POINT = 'highest_point', 'Highest Point'
    GDP = 'gdp', 'GDP'
    LIFE_EXPECTANCY = 'life_expectancy', 'Life Expectancy'
    # FUTURE: Add more categories by just adding them here
    # BORDER_COUNTRIES = 'border_countries', 'Bordering Countries'
    # FAMOUS_LANDMARKS = 'famous_landmarks', 'Famous Landmarks'

# Extensible format choices - add new answer formats here
class QuestionFormat(models.TextChoices):
```

```
    TEXT_INPUT = 'text_input', 'Text Input'
    MULTIPLE_CHOICE = 'multiple_choice', 'Multiple Choice'
    TRUE_FALSE = 'true_false', 'True/False'
    # FUTURE: Add more formats by just adding them here
    # MAP_LOCATION = 'map_location', 'Locate on Map'
    # DRAG_DROP = 'drag_drop', 'Drag and Drop'
    # IMAGE_SELECT = 'image_select', 'Select Image'

class Question(Model):
    """
    Flexible question model supporting multiple categories and answer
formats.
    New categories/formats can be added without migrations - just
update choices above.
    """
    # What aspect of the country are we asking about?
    category = CharField(
        max_length=30,
        choices=QuestionCategory.choices
    )

    # How should the user answer?
    format = CharField(
        max_length=30,
        choices=QuestionFormat.choices
    )

    # Which country is this about?
    country = ForeignKey(Country, on_delete=models.CASCADE)

    # The actual question text
    question_text = TextField()  # "What is the capital of France?"

    # Correct answer (structure varies by format, stored as JSON)
    # Examples:
    #   TEXT_INPUT: {"answer": "Paris", "alternates": ["paris",
"París"]}
    #   MULTIPLE_CHOICE: {"correct": "Paris", "options": ["Paris",
"London", "Berlin", "Madrid"]}
    #   TRUE_FALSE: {"answer": true}
```

```
    #   MAP_LOCATION: {"latitude": 48.8566, "longitude": 2.3522,
"tolerance_km": 50}
    correct_answer = JSONField()

    # Additional metadata for generating/displaying questions
    # Examples:
    #   - difficulty_override: manual difficulty for this specific
question
    #   - hint: optional hint text
    #   - explanation: why this answer is correct
    #   - image_url: for questions that show an image
    metadata = JSONField(default=dict, blank=True)

    # For daily challenges
    daily_challenge = ForeignKey('DailyChallenge', null=True,
blank=True, on_delete=models.SET_NULL)

    # For quiz mode - groups questions into sessions
    quiz_session = ForeignKey('QuizSession', null=True, blank=True,
on_delete=models.CASCADE)

    # Tracking
    created_at = DateTimeField(auto_now_add=True)

    class Meta:
        indexes = [
            Index(fields=['category', 'format']),  # For filtering
question types
            Index(fields=['country']),  # For country-specific queries
            Index(fields=['daily_challenge']),  # For daily challenge
lookups
            Index(fields=['quiz_session']),  # For quiz result grouping
        ]

    def __str__(self):
        return f"{self.category} - {self.format} - {self.country.name}"

    def validate_answer(self, user_answer_data):
        """
        Validate user's answer against correct answer.
        Returns (is_correct: bool, explanation: str)
```

```
        """
        if self.format == QuestionFormat.TEXT_INPUT:
            user_text = user_answer_data.get('text',
'').lower().strip()
            correct = self.correct_answer['answer'].lower()
            alternates = [alt.lower() for alt in
self.correct_answer.get('alternates', [])]

            is_correct = user_text == correct or user_text in
alternates
            explanation = f"Correct answer:
{self.correct_answer['answer']}"
            return is_correct, explanation

        elif self.format == QuestionFormat.MULTIPLE_CHOICE:
            user_choice = user_answer_data.get('selected_option')
            is_correct = user_choice == self.correct_answer['correct']
            explanation = f"Correct answer:
{self.correct_answer['correct']}"
            return is_correct, explanation

        elif self.format == QuestionFormat.TRUE_FALSE:
            user_bool = user_answer_data.get('answer')
            is_correct = user_bool == self.correct_answer['answer']
            explanation = f"The statement is
{self.correct_answer['answer']}"
            return is_correct, explanation

        # FUTURE: Add validation for new formats here
        # elif self.format == QuestionFormat.MAP_LOCATION:
        #     user_lat = user_answer_data.get('latitude')
        #     user_lng = user_answer_data.get('longitude')
        #     correct_lat = self.correct_answer['latitude']
        #     correct_lng = self.correct_answer['longitude']
        #     tolerance = self.correct_answer.get('tolerance_km', 100)
        #
        #     distance = calculate_distance(user_lat, user_lng,
correct_lat, correct_lng)
        #     is_correct = distance <= tolerance
        #     explanation = f"Correct location: {correct_lat},
{correct_lng}"
```

```
        #     return is_correct, explanation

        return False, "Unknown question format"


#### UserAnswer Model
```python
class UserAnswer(Model):
    """
    Records a user's answer to a question.
    Flexible structure supports any answer format via JSON.
    """
    user = ForeignKey(User, on_delete=models.CASCADE)
    question = ForeignKey(Question, on_delete=models.CASCADE)

    # User's answer (structure varies by question format, stored as
JSON)
    # Examples:
    #   TEXT_INPUT: {"text": "Paris"}
    #   MULTIPLE_CHOICE: {"selected_option": "Paris"}
    #   TRUE_FALSE: {"answer": true}
    #   MAP_LOCATION: {"latitude": 48.85, "longitude": 2.35}
    answer_data = JSONField()

    # Result
    is_correct = BooleanField()
    explanation = TextField(blank=True)  # Why answer was
correct/incorrect

    # Attempt tracking (for daily challenge's 3-attempt limit)
    attempt_number = IntegerField(default=1)  # 1, 2, or 3

    # Performance tracking
    time_taken_seconds = IntegerField(null=True, blank=True)

    # Timestamps
    answered_at = DateTimeField(auto_now_add=True)

    class Meta:
        indexes = [
            Index(fields=['user', 'question']),
```
```

```
            Index(fields=['user', 'is_correct']),  # For stats queries
            Index(fields=['question']),
        ]
        ordering = ['-answered_at']

    def __str__(self):
        status = "✓" if self.is_correct else "✗"
        return f"{status} {self.user.username} - {self.question}"


#### QuizSession Model (Phase 3)
```python
class QuizSession(Model):
    """
    Groups questions together for quiz mode.
    Daily challenges don't use sessions (single question per day).
    """
    user = ForeignKey(User, on_delete=models.CASCADE)

    # Quiz configuration
    question_count = IntegerField()
    selected_categories = JSONField()  # ["flag", "capital",
"population"]
    selected_formats = JSONField()  # ["text_input", "multiple_choice"]
or null for any
    difficulty_filter = CharField(max_length=10, null=True, blank=True)
    region_filter = CharField(max_length=50, null=True, blank=True)

    # Quiz type/mode
    quiz_type = CharField(
        max_length=30,
        choices=[
            ('custom', 'Custom Quiz'),
            ('speed', 'Speed Round'),
            ('regional', 'Regional Quiz'),
            ('category_master', 'Category Master'),
        ],
        default='custom'
    )

    # Performance
```
```

```
    score = IntegerField(default=0)  # Number correct
    total_time_seconds = IntegerField(null=True, blank=True)

    # Status
    is_completed = BooleanField(default=False)

    # Timestamps
    started_at = DateTimeField(auto_now_add=True)
    completed_at = DateTimeField(null=True, blank=True)

    class Meta:
        ordering = ['-started_at']

    def __str__(self):
        return f"{self.user.username} - {self.quiz_type} -
{self.score}/{self.question_count}"
```

Design Benefits:

1. **Easy Category Addition:** Just add to `QuestionCategory` enum - no migration needed
2. **Easy Format Addition:** Just add to `QuestionFormat` enum and add validation logic in `validate_answer()`
3. **Flexible Answer Storage:** JSON fields adapt to any answer structure
4. **User Format Control:** `QuizSession.selected_formats` lets users choose which formats they want
5. **Consistent Validation:** Central `validate_answer()` method handles all formats
6. **Future-Proof:** Map-based questions, drag-drop, image selection all supported by design

Migration Path from MVP:

- MVP: Use `Question` model with only `TEXT_INPUT` format and `FLAG` category for daily challenges
- Phase 3: Activate multiple categories and formats for quiz mode
- No breaking changes - just add more question generation logic

Future Models (Designed but Not Implemented)

DifficultyTierState (Phase 2)

```
class DifficultyTierState(Model):
    """Tracks which countries have been shown in each tier's cycle"""
    tier = CharField(max_length=10)  # 'easy', 'medium', 'hard'
    cycle_number = IntegerField(default=1)
    shown_countries = JSONField(default=list)  # List of country codes
    cycle_start_date = DateField()
    last_shown_date = DateField()
```

Achievement (Phase 3)

```
class Achievement(Model):
    """Future gamification system"""
    code = CharField(max_length=50, unique=True)
    name = CharField(max_length=100)
    description = TextField()
    icon = CharField(max_length=50)
    requirement = JSONField()  # Flexible achievement criteria
```

UserAchievement (Phase 3)

```
class UserAchievement(Model):
    user = ForeignKey(User)
    achievement = ForeignKey(Achievement)
    unlocked_at = DateTimeField(auto_now_add=True)
```

MVP Feature Set

Phase 1: MVP (Weeks 1-5)

1. Backend API (Week 1)

Goals:

- Set up Django project with two apps (`users`, `flags`)
- Implement OAuth authentication with Google
- Cache REST Countries API data in PostgreSQL
- Build daily flag selection algorithm (simple random, no repeats)
- Create RESTful endpoints

Endpoints:

```
Authentication:
POST   /api/auth/google/           # OAuth login
GET    /api/auth/user/             # Get current user
POST   /api/auth/logout/           # Logout

Daily Challenge:
GET    /api/daily/                 # Get today's challenge (returns
Question object)
POST   /api/daily/answer/          # Submit an answer (creates
UserAnswer)
GET    /api/daily/result/          # Get today's result (after
answering)
GET    /api/daily/history/         # Get past daily challenges

User Stats:
GET    /api/stats/                 # Get user statistics
GET    /api/stats/categories/      # Get per-category breakdown (Phase
3)
GET    /api/stats/formats/         # Get per-format breakdown (Phase 3)

Quiz Mode (Phase 3):
POST   /api/quiz/create/           # Create quiz session with config
GET    /api/quiz/{session_id}/     # Get quiz questions
POST   /api/quiz/{session_id}/answer/  # Submit answer to quiz question
GET    /api/quiz/{session_id}/results/ # Get quiz results
GET    /api/quiz/history/          # Get past quiz sessions
```

```
Encyclopedia:
GET    /api/countries/             # List all countries
GET    /api/countries/search/      # Search by name or flag emoji
GET    /api/countries/{code}/      # Get country details

Question Generation (Internal/Admin):
POST   /api/admin/questions/generate/  # Generate questions for
countries
GET    /api/admin/questions/           # List all questions
```

Success Criteria:

- All endpoints return correct data
- Daily flag is deterministic (same for all users)
- Resets at midnight Eastern Time
- No repeated countries until all shown

2. React Website (Weeks 2-3)

Goals:

- Set up React app with MUI styling
- Implement Zustand state management
- Build all core screens
- Integrate Leaflet maps

Pages:

1. **Landing Page** (unauthenticated)
    - Hero section with app description
    - Google Sign-In button
    - Preview of daily challenge
2. **Daily Challenge Page** (authenticated)
    - Flag display (emoji only, large)
    - Text input for country guess

- Submit button
- Attempts counter (X/3)
- Running stats display (correct count, current streak)

3. **Country Info Page** (after guess or from encyclopedia)
   - All country data fields listed in requirements
   - Interactive Leaflet map (zoom + pan)
   - Static map view
   - "Back to Today's Challenge" button
4. **Encyclopedia Page**
   - Search bar (by country name or flag emoji)
   - Results grid with flag emoji + country name
   - Click country → shows full country info
5. **Profile/Stats Page**
   - Total correct guesses
   - Current streak
   - Longest streak
   - List of incorrectly guessed countries
   - Logout button

Success Criteria:

- Smooth user flow from login → guess → result
- Stats update in real-time
- Search works reliably
- Maps display correctly
- Mobile-responsive design

3. iOS App (Week 4)

Goals:

- Set up React Native with Expo

- Reuse Zustand store from web
- Adapt MUI components to React Native Paper
- Integrate React Native Maps

Screens:

- Mirror web pages but optimized for mobile
- Native navigation patterns
- Tab bar navigation (Today / Encyclopedia / Profile)

Success Criteria:

- Feature parity with web
- Smooth navigation
- Works on iOS simulator and device
- Maps render correctly

4. iOS Widget (Week 5)

Goals:

- Build native WidgetKit widget in Swift
- Fetch today's flag from API
- Deep link to app on tap

Widget Features:

- Small/Medium/Large sizes
- Shows flag emoji only
- Shows "Tap to guess" text
- Updates at midnight ET
- Opens app to guess screen on tap

Success Criteria:

- Widget updates daily

- Deep linking works
- Handles offline gracefully

Future Roadmap

Phase 2: Enhanced Difficulty System (Weeks 6-7)

New Features:

- Three difficulty tiers (Easy, Medium, Hard)
- Recognition-based categorization
- Day-of-week rotation (Mon=Easy, Tue=Med, Wed=Hard, etc.)
- Independent tier cycles (each tier shuffles separately)
- Admin panel for recategorizing countries

Backend Changes:

- Implement `DifficultyTierState` model
- Update daily flag algorithm to use tiers
- Create tier management endpoints
- Build tier categorization tool (analyze country popularity)

Frontend Changes:

- Display difficulty badge on daily challenge
- Show tier progress (X/65 countries in Easy tier)
- Difficulty filter in encyclopedia

Timeline: 1-2 weeks (backend algorithm is complex)

Phase 3: Quiz Mode (Weeks 8-10)

New Features:

- **Flexible Question Categories:**
  - Flag recognition (current daily challenge style)
  - Population ("Which country has X population?")
  - Capital ("What is the capital of X?")
  - Largest City ("What is the most populous city in X?")
  - Languages ("Which language(s) are spoken in X?")
  - Currency ("What currency does X use?")
  - Area ("Which country is larger: X or Y?")
  - Geography ("Which continent is X located in?")
  - Mix & match multiple categories in one quiz
- **Quiz Configuration:**
  - Choose question count (5, 10, 15, 20, 25 questions)
  - Select specific category or multiple categories
  - Random country selection within chosen parameters
  - Optional: Regional focus (e.g., "African capitals only")
  - Optional: Difficulty filter (easy/medium/hard countries)
- **Quiz Types:**
  - Custom Quiz (user-configured categories)
  - Speed Round (10 questions, 30 seconds per question)
  - Regional Quiz (all questions about one region)
  - Category Master (all questions from one category)
- **Quiz Stats & History:**
  - Per-category accuracy (% correct for capitals, flags, etc.)
  - Time-based stats (average time per question)
  - Category mastery progress
  - Quiz history with detailed results

Backend Changes:

- New models: `QuizSession`, `QuizQuestion`, `QuizAnswer`

- `QuestionType` enum/choices for all categories
- Quiz generation engine (random country + category selection)
- Answer validation for each question type
- Quiz statistics aggregation
- Optional: Leaderboard endpoints

Frontend Changes:

- Quiz configuration screen (select categories, count, options)
- Active quiz screen (dynamic question rendering per type)
- Progress indicator (X/10 questions, timer if applicable)
- Results screen with breakdown by category
- Quiz history page with filters
- Category mastery dashboard

Data Model Preview (Phase 3):

```
class QuizSession(Model):
    user = ForeignKey(User)
    question_count = IntegerField()
    categories = JSONField()  # ["flag", "capital", "population"]
    difficulty_filter = CharField(null=True)
    region_filter = CharField(null=True)
    is_timed = BooleanField(default=False)
    created_at = DateTimeField(auto_now_add=True)
    completed_at = DateTimeField(null=True)

class QuizQuestion(Model):
    session = ForeignKey(QuizSession)
    country = ForeignKey(Country)
    question_type = CharField(choices=QUESTION_TYPES)
    question_text = TextField()  # "What is the capital of France?"
    correct_answer = TextField()
    user_answer = TextField(null=True)
    is_correct = BooleanField(null=True)
    time_taken_seconds = IntegerField(null=True)
```

Timeline: 2-3 weeks (significant new functionality, complex question generation)

Phase 4: Social & Gamification (Weeks 11-13)

New Features:

- Achievements system (50+ achievements)
- Share results to social media
- Friend challenges
- Custom avatar/profile
- Badge collection

Backend Changes:

- `Achievement`, `UserAchievement` models
- Social sharing metadata generation
- Friend system (optional)

Frontend Changes:

- Achievements gallery
- Share result UI
- Profile customization
- Badge display on profile

Timeline: 2-3 weeks

Phase 5: Content Expansion (Ongoing)

New Features:

- Historical facts per country
- Famous landmarks
- Cultural trivia

- Regional groupings (EU, ASEAN, etc.)
- Custom curated collections

Backend Changes:

- Extended `CountryMetadata` model
- Content management system
- Editorial workflow (if you add editors)

Frontend Changes:

- Enhanced country info pages
- Collection browsing
- Daily trivia alongside flag

Timeline: Ongoing / As desired

Development Timeline

Week 1: Backend Foundation

Days 1-2: Django Setup

- Install Python, Django, PostgreSQL (for local dev use SQLite)
- Create project structure
- Set up two Django apps (`users`, `flags`)
- Configure Django REST Framework
- Set up `django-allauth` for OAuth

Days 3-4: Authentication & Models

- Configure Google OAuth
- Create all database models
- Run migrations
- Write model tests

Days 5-7: API Endpoints & Algorithm

- Build daily flag selection algorithm
- Cache REST Countries API data
- Implement all MVP endpoints
- Test API with Postman/curl
- Write API documentation

Milestones:

- ✅ Can authenticate with Google
- ✅ Daily flag endpoint returns same country for all users
- ✅ Guess submission works correctly
- ✅ Stats update properly

Week 2: React Website (Part 1)

Days 8-9: Project Setup

- Create React app
- Install MUI and Zustand
- Set up routing
- Configure Axios for API calls
- Create environment variables

Days 10-11: Authentication Flow

- Build landing page
- Implement Google OAuth flow (frontend)
- Create authenticated layout
- Handle token storage and refresh

Days 12-14: Daily Challenge

- Build daily challenge page
- Implement guess input and submission
- Show attempts counter
- Display user stats
- Handle correct/incorrect feedback
- Create country info display

Milestones:

- ✅ Can log in from website
- ✅ Can see today's flag and submit guesses
- ✅ Stats display correctly

Week 3: React Website (Part 2)

Days 15-17: Encyclopedia & Maps

- Build encyclopedia search page
- Implement search functionality
- Integrate Leaflet maps
- Create static map component
- Polish country info page

Days 18-19: Profile & Stats

- Build profile page
- Display all user statistics
- Show incorrect countries list
- Add logout functionality

Days 20-21: Testing & Polish

- Responsive design testing
- Cross-browser testing

- Fix bugs
- Polish UI/UX
- Add loading states and error handling

Milestones:

- ✅ Website fully functional
- ✅ Mobile-responsive
- ✅ All features working smoothly

Week 4: iOS App

Days 22-23: React Native Setup

- Create Expo project
- Install React Native Paper
- Set up navigation (tabs + stack)
- Configure API client (reuse Axios setup)

Days 24-26: Core Features

- Port authentication flow
- Build daily challenge screen
- Implement encyclopedia
- Add profile page
- Integrate React Native Maps

Days 27-28: Testing & Polish

- Test on iOS simulator
- Test on physical device
- Handle edge cases
- Polish navigation and UX

Milestones:

- ✅ App feature parity with website
- ✅ Smooth native experience

Week 5: iOS Widget & Deployment

Days 29-31: Widget Development

- Learn WidgetKit basics (Swift)
- Create widget extension
- Fetch today's flag from API
- Implement deep linking
- Test widget timeline updates

Days 32-33: Deployment Preparation

- Choose hosting platform (Railway/Render/AWS)
- Set up production database (PostgreSQL)
- Configure environment variables
- Set up SSL/HTTPS
- Test production API

Days 34-35: Final Testing & Launch

- End-to-end testing
- Performance optimization
- App Store submission prep (screenshots, description)
- Submit to App Store
- Launch website

Milestones:

- ✅ Widget working and updating daily

- ✅ Backend deployed to production
- ✅ Website live
- ✅ App submitted to App Store

Technical Decisions Log

Locked Decisions

Decision	Choice	Rationale
Backend Framework	Django + DRF	Learning goal, robust, great for rapid development
Web Frontend	React + MUI	Component reuse, consistent design, learning goal
Mobile Framework	React Native + Expo	Code sharing with web, faster than native
Widget	Native WidgetKit	Expo doesn't support widgets, must be native
State Management	Zustand	Simple, works cross-platform, easy to learn
Database (dev)	SQLite	Fast setup, no config needed
Database (prod)	PostgreSQL	Industry standard, JSON support, robust
OAuth Provider	Google	Universal, simple integration, MVP-sufficient
Maps Library	Leaflet (web)	Free, simple, sufficient for MVP
Daily Reset Time	Midnight Eastern Time	Fixed timezone, consistent globally
Flag Selection (MVP)	Random, no repeats	Simple to implement, proves concept

Deferred Decisions

Decision	Timeline	Notes
Production Hosting	Week 3-4	Railway vs AWS, decided at deployment
Mobile Maps	Week 4	React Native Maps or Mapbox Native
Difficulty Categorization	Phase 2	Need to analyze country recognition data
Additional OAuth	Phase 2+	Apple required for App Store if adding social logins
Error Handling Strategy	After Week 1	Starting with Django REST Framework defaults. Will evaluate need for custom exception classes after building initial endpoints.

Architecture Guardrails

Extensibility Checklist

To ensure we don't paint ourselves into a corner, we follow these principles:

✅ Database Design

- All models include `created_at` and `updated_at` where relevant
- Use JSONField for future-flexible data (avoid migrations)
- Add indexed fields for future queries even if unused in MVP
- Nullable foreign keys where relationships might expand

✅ API Design

- Version API endpoints (`/api/v1/...`) for future breaking changes
- Include pagination on all list endpoints (even if small now)
- Return full objects, not minimal data (easier to extend UI)
- Use consistent response structure:

```
{
  "data": {...},
  "meta": {...},
  "errors": null
}
```

✅ Frontend Architecture

- Separate API calls into service layer (not in components)
- Use feature-based folder structure, not type-based:

```
/features
  /dailyChallenge
    - DailyChallengeScreen.jsx
    - useDailyChallenge.js
    - dailyChallengeService.js
  /encyclopedia
    - EncyclopediaScreen.jsx
    - ...
```

- Extract repeated logic into custom hooks
- Keep component state local; only global state in Zustand

✅ State Management

- Separate concerns in Zustand store:

```
// Good: Separate slices
const useAuthStore = create(...)
const useFlagStore = create(...)
const useStatsStore = create(...)

// Bad: One massive store
const useStore = create(...)
```

- Write selectors for complex state access
- Keep actions pure (side effects in services)

✅ Testing Strategy

- Write tests for business logic (flag algorithm, stats calculation)
- Don't test UI in MVP (slows development)
- Test API endpoints with Django's test suite
- Manual testing for UX flows

Risk Assessment & Mitigation

Technical Risks

Risk 1: React Native + WidgetKit Integration

- **Likelihood:** High
- **Impact:** Medium
- **Mitigation:** Widget is separate Swift project; communicates via API and deep links only
- **Fallback:** Launch without widget initially

Risk 2: Daily Flag Algorithm Complexity

- **Likelihood:** Medium (for Phase 2 difficulty tiers)
- **Impact:** Medium
- **Mitigation:** Build simple version in MVP; validate with users before adding tiers
- **Fallback:** Keep simple random algorithm

Risk 3: App Store Rejection

- **Likelihood:** Low
- **Impact:** High (delays launch)
- **Mitigation:** Follow Apple guidelines strictly; prepare privacy policy; test thoroughly
- **Fallback:** Launch web-only first, iterate on app

Risk 4: REST Countries API Downtime

- **Likelihood:** Low
- **Impact:** High (no data source)
- **Mitigation:** Cache aggressively; store all data in our database; refresh weekly not daily
- **Fallback:** Manual data entry (one-time, manageable for 195 countries)

Timeline Risks

Risk 1: Learning Curve Longer Than Expected

- **Likelihood:** Medium
- **Impact:** Medium
- **Mitigation:** Focus on MVP features only; use tutorials/docs heavily; ask for help
- **Contingency:** Add 1-2 weeks if needed

Risk 2: Scope Creep

- **Likelihood:** High (common in solo projects)
- **Impact:** High (delays launch)
- **Mitigation:** Strict MVP focus; resist adding "small" features; defer to Phase 2
- **Commitment:** No new features until MVP launches

Success Metrics

MVP Launch Criteria

- [ ] User can authenticate with Google
- [ ] User sees a new flag every day (same as other users)
- [ ] User can guess with 3 attempts
- [ ] Stats track correctly (correct count, streaks, incorrect countries)
- [ ] Encyclopedia search works
- [ ] All country data displays properly

- [ ] Maps render interactively
- [ ] Website is responsive (mobile + desktop)
- [ ] iOS app has feature parity with website
- [ ] iOS widget displays today's flag and deep links to app
- [ ] Backend is deployed to production
- [ ] App is submitted to App Store

Post-Launch Metrics (if adding analytics)

- Daily Active Users (DAU)
- Streak retention (% of users maintaining 7+ day streak)
- Average guesses per flag (lower = easier flags)
- Encyclopedia usage rate
- Completion rate (% who guess vs abandon)

Open Questions

~~Immediate (Week 1)~~ - RESOLVED

- [x] **REST Countries API caching strategy:** Store selected fields explicitly in schema (Country model fields) + full JSON in `raw_api_response` field for flexibility
- [x] **Token refresh strategy:** Access tokens valid for 1 day, refresh tokens valid for 30 days (industry standard for web apps). Django OAuth handles refresh automatically via django-allauth.
- [x] **Error handling approach:** Start with Django REST Framework's default exception handling. Will evaluate need for custom exceptions after building first endpoints. (See Deferred Decisions below)

Before Phase 2

- [ ] How to categorize countries by "recognition" (data source? manual?)
- [ ] Should we add a tier preview/legend for users?

- [ ] Do we want admin UI for tier management? (Django Admin vs custom?)

Before Phase 3

- [ ] Quiz question generation algorithm: How to ensure variety and appropriate difficulty?
- [ ] Answer validation for open-ended questions (e.g., accepting "USA" vs "United States")?
- [ ] Leaderboards: global, friends-only, or both?
- [ ] Do we want real-time multiplayer quizzes?
- [ ] Time limits per question by category? (capitals might be faster than population)

Before Phase 4

- [ ] Achievement criteria: manual curation or algorithm-generated?
- [ ] Social sharing: which platforms to support? (Twitter, Instagram, iMessage?)

Before App Store Launch

- [ ] Privacy policy (can use template, but customize for our data usage)
- [ ] App Store screenshots strategy (hire designer or DIY?)
- [ ] Pricing model (free, freemium, paid?) - assuming free for MVP

Appendix

Learning Resources

Django:

- Official Tutorial: https://docs.djangoproject.com/en/5.0/intro/tutorial01/
- Django REST Framework: https://www.django-rest-framework.org/tutorial/quickstart/
- django-allauth: https://django-allauth.readthedocs.io/

React:

- Official Docs: https://react.dev/learn
- Material-UI: https://mui.com/material-ui/getting-started/
- Zustand: https://github.com/pmndrs/zustand

React Native:

- Expo Docs: https://docs.expo.dev/
- React Native Paper: https://callstack.github.io/react-native-paper/
- React Navigation: https://reactnavigation.org/

iOS Widget:

- WidgetKit: https://developer.apple.com/documentation/widgetkit
- Swift Tutorial: https://docs.swift.org/swift-book/

API Documentation

REST Countries API:

- Base URL: https://restcountries.com/v3.1/
- All countries: `/all`
- By name: `/name/{name}`
- By code: `/alpha/{code}`
- Fields parameter: `?fields=name,capital,population,flag`

Data Fields We'll Use:

- name.common, name.official
- capital (array)
- population
- area
- languages (object)
- currencies (object)
- flags.emoji

- latlng (array)
- region, subregion
- timezones (array)

Deployment Checklist

Pre-Deployment:

- [ ] Environment variables configured
- [ ] Database migrations run
- [ ] Static files collected
- [ ] SECRET_KEY set to secure random value
- [ ] DEBUG=False in production
- [ ] ALLOWED_HOSTS configured
- [ ] CORS settings configured for frontend domains

Post-Deployment:

- [ ] SSL certificate installed (HTTPS)
- [ ] Custom domain configured (if applicable)
- [ ] Monitoring set up (errors, uptime)
- [ ] Backups configured (database)
- [ ] Rate limiting enabled (prevent abuse)

Document History

Version	Date	Changes	Author
1.0	Oct 22, 2025	Initial design document	Solo Developer
1.1	Oct 22, 2025	**Refinements:** (1) Clarified MUI vs React Native Paper code reusability expectations (~30% rewrite UI, ~70% shared logic); (2) Expanded Phase 3 quiz vision to include multiple question categories (capital, population, currency, etc.) and flexible quiz configuration; (3) Enhanced data models for extensibility (`GuessAttempt`, `Country`, `UserStats`) to support future quiz categories without migrations; (4) Resolved Phase 1 open questions (API caching strategy, token refresh, error handling); (5) Added error handling to deferred decisions	Solo Developer
1.2	Oct 22, 2025	**Major Redesign:** Completely rethought question/answer system for maximum flexibility: (1) Replaced `GuessAttempt` with `Question` + `UserAnswer` models supporting multiple answer formats (text input, multiple choice, true/false, future map-based); (2) Added `QuestionCategory` and `QuestionFormat` enums for easy extension without migrations; (3) Implemented flexible answer validation system; (4) Added `QuizSession` model for Phase 3 quiz configuration; (5) Enhanced `UserStats` to track performance by both category AND format; (6) Added comprehensive "Question Generation System" section with step-by-step guides for adding categories/formats; (7) Updated API endpoints to reflect new architecture; (8) Added quiz configuration system allowing users to select answer formats	Solo Developer

Notes for Future Self

Key Lessons to Remember:

1. **Don't optimize prematurely** - Get it working first, optimize later
2. **User feedback > assumptions** - Launch MVP, see what users actually want
3. **Code reuse has limits** - Don't force sharing between web/mobile if it adds complexity
4. **Document as you go** - Future you will thank present you
5. **Test the happy path first** - Edge cases can wait until MVP works

When You're Stuck:

- Check this document's architecture section
- Review the extensibility checklist
- Remember: MVP first, features later
- Ask for help (forums, Discord, Claude)

When You Want to Add a Feature:

- Ask: "Does this block MVP launch?"
- If no → add to Phase 2+
- If yes → was it in original requirements?
- Be ruthless about scope

Contact & Support

Developer: Solo Project (You!)
Project Repository: [To be created]
Issue Tracking: [To be created]
Design Feedback: [To be determined]

Remember: This is your roadmap. It will evolve as you learn and build. That's expected and healthy. The goal isn't to follow this perfectly—it's to have a north star that keeps you aligned and prevents feature creep while staying flexible for growth.

Now let's build this! 🚀