Code Review Package Documentation

Abroadly - Backend REST API Subsystem

1. Team Members

- Lucas Slater
- Gordon Song
- Tae Kim
- Trey Fisher

2. Project Name

Abroadly - A peer-verified study abroad platform

3. Subsystem for Review

Backend REST API Layer

This subsystem includes:

- Database Models (`app/models.py`)
- Programs API (`app/programs/routes.py`)
- Places API (`app/places/routes.py`)
- Trips API (`app/trips/routes.py`)

Total Lines of Code: ~693 lines (excluding authentication routes)

4. System Context

4.1 System Overview

Abroadly is a comprehensive study abroad platform that helps students in three ways:

1. **For Prospective Students:** Browse and review study abroad programs (institutions, costs, housing, courses)
2. **For Current Students:** Discover and review local places (restaurants, activities, museums, housing in their host city)
3. **For Weekend Travelers:** Plan and review weekend trips to nearby destinations

4.2 Architecture Overview

```
┌─────────────────────────────────────────────────────┐
│          React Frontend (Vite + TypeScript)          │
│  - Home, Programs, Places, Trips pages               │
│  - API service layer                                 │
└───────────────────┬─────────────────────────────────┘
                    │ HTTP/REST
                    ▼
┌─────────────────────────────────────────────────────┐
│         FastAPI Backend (Python)                     │
│  ┌─────────────────────────────────────────────┐   │
│  │  SUBSYSTEM UNDER REVIEW                     │   │
│  │  ┌─────────────────────────────────────┐    │   │
│  │  │  Database Models (SQLModel)         │    │   │
│  │  │  - StudyAbroadProgram, Place, Trip  │    │   │
│  │  │  - Reviews for each entity          │    │   │
│  │  └─────────────────────────────────────┘    │   │
│  │  ┌─────────────────────────────────────┐    │   │
│  │  │  REST API Routes                    │    │   │
│  │  │  - /api/programs/  (CRUD + reviews)│    │   │
│  │  │  - /api/places/    (CRUD + reviews)│    │   │
│  │  │  - /api/trips/     (CRUD + reviews)│    │   │
│  │  └─────────────────────────────────────┘    │   │
│  └─────────────────────────────────────────────┘   │
│                                                      │
│  Other Components (not in review):                  │
│  - Authentication (magic link + JWT)                │
│  - Database connection management                   │
│  - Dependency injection                             │
└───────────────────┬─────────────────────────────────┘
                    │
                    ▼
           ┌───────────────┐
           │ SQLite Database│
           └───────────────┘
```

4.3 How This Subsystem Fits

The REST API layer is the **core business logic** of the backend. It:

- Defines the data structures (models) for programs, places, and trips
- Implements CRUD operations for all entities
- Handles reviews for each entity type
- Enforces authentication on write operations
- Provides filtering and pagination for list endpoints

This subsystem is called by the React frontend through HTTP requests and interacts with the database through SQLModel ORM.

---

5. Informal Specification

5.1 User Stories Implemented

Story 1: Browse Study Abroad Programs

> As a prospective study abroad student, I want to browse and search study abroad programs by city and country, so I can find programs that match my preferences.

Story 2: Review Programs

> As a study abroad student, I want to leave detailed reviews about programs (overall experience, courses, and housing), so I can help future students make informed decisions.

Story 3: Discover Local Places

> As a current study abroad student, I want to discover and review local places (restaurants, activities, housing) in my host city, categorized by type.

Story 4: Plan Weekend Trips

> As a study abroad student, I want to browse and review weekend trip destinations shared by other students.

Story 5: CRUD Operations (Admin/Power Users)

> As an authenticated user, I want to create, update, and delete programs, places, and trips.

5.2 Data Model

Core Entities

StudyAbroadProgram

- `program_name` : Name of the program (e.g., "Vanderbilt in Barcelona")
- `institution` : Home institution
- `city` , `country` : Location
- `cost` : Program cost
- `housing_type` : Type of housing provided
- `duration` : Length of program
- `description` : Detailed description

Place

- `name` : Name of the place
- `category` : Type (restaurant, activity, museum, housing, nightlife, etc.)
- `city` , `country` : Location
- `latitude` , `longitude` : GPS coordinates for map integration
- `address` : Street address
- `description` : Details about the place

Trip

- `destination` : Trip destination
- `country` : Country
- `trip_type` : Type (weekend, spring break, summer, etc.)
- `description` : Trip details

Review Entities

Each core entity has associated reviews:

- **ProgramReview**: Overall program rating and review
- **CourseReview**: Specific course ratings (tied to programs)
- **ProgramHousingReview**: Housing experience reviews (tied to programs)
- **PlaceReview**: Place ratings and reviews
- **TripReview**: Trip ratings and reviews

All reviews include:

- `user_id` : Foreign key to User (who wrote it)
- `rating` : 1-5 stars (validated at model level)
- `review_text` : Text content
- `date` : Timestamp

5.3 API Endpoints

Programs API (`/api/programs/`)

Method	Endpoint	Auth Required	Description
GET	`/api/programs/`	No	List programs (filter by city, country; pagination)
GET	`/api/programs/{id}`	No	Get specific program
POST	`/api/programs/`	Yes	Create new program
PUT	`/api/programs/{id}`	Yes	Update program
DELETE	`/api/programs/{id}`	Yes	Delete program
POST	`/api/programs/{id}/reviews`	Yes	Add program review
GET	`/api/programs/{id}/reviews`	No	List program reviews
POST	`/api/programs/{id}/courses/reviews`	Yes	Add course review
GET	`/api/programs/{id}/courses/reviews`	No	List course reviews

Method	Endpoint	Auth Required	Description
POST	`/api/programs/{id}/housing/reviews`	Yes	Add housing review
GET	`/api/programs/{id}/housing/reviews`	No	List housing reviews

Places API (`/api/places/`)

Method	Endpoint	Auth Required	Description
GET	`/api/places/`	No	List places (filter by city, country, category)
GET	`/api/places/{id}`	No	Get specific place
POST	`/api/places/`	Yes	Create new place
PUT	`/api/places/{id}`	Yes	Update place
DELETE	`/api/places/{id}`	Yes	Delete place
POST	`/api/places/{id}/reviews`	Yes	Add place review
GET	`/api/places/{id}/reviews`	No	List place reviews

Trips API (`/api/trips/`)

Method	Endpoint	Auth Required	Description
GET	`/api/trips/`	No	List trips (filter by destination, country, trip_type)
GET	`/api/trips/{id}`	No	Get specific trip
POST	`/api/trips/`	Yes	Create new trip
PUT	`/api/trips/{id}`	Yes	Update trip
DELETE	`/api/trips/{id}`	Yes	Delete trip
POST	`/api/trips/{id}/reviews`	Yes	Add trip review
GET	`/api/trips/{id}/reviews`	No	List trip reviews

5.4 Key Design Patterns

1. Repository Pattern

- SQLModel ORM handles database operations
- Session dependency injection via `Depends(get_session)`

2. Authentication Middleware

- Write operations require `user: User = Depends(current_user)`
- Read operations are public (no authentication required)

3. Pydantic Schemas

- Separate create/update schemas for input validation
- `exclude_unset=True` on updates allows partial updates

4. RESTful Design

- Standard HTTP status codes (201 Created, 204 No Content, 404 Not Found)
- Resource-oriented URLs
- Consistent response formats

5. Error Handling

- HTTPException with appropriate status codes
- Existence checks before updates/deletes
- Foreign key validation for reviews

5.5 Algorithms & Data Structures

Filtering Algorithm (used in all list endpoints):

```
query = select(Model)
if filter_param:
    query = query.where(Model.field == filter_param)
query = query.offset(skip).limit(limit)
results = session.exec(query).all()
```

Partial Update Algorithm:

```
update_data = schema.model_dump(exclude_unset=True)
for key, value in update_data.items():
    setattr(instance, key, value)
```

Data Structures:

- **Database:** SQLite with SQLModel ORM (relational tables with foreign keys)
- **Indexes:** Created on searchable fields (city, country, category, program_name)
- **Pagination:** Offset/limit pattern for scalability

---

6. Most Likely Changes (Optional)

Based on the current implementation, here are the most likely future changes:

6.1 Search Enhancement

Current: Exact match filtering only (`where(field == value)`)
Likely Change: Add fuzzy search, full-text search for names/descriptions
Impact: Would need to modify `list_*` endpoints to accept search queries and use SQL LIKE or search engine

6.2 Aggregated Ratings

Current: Reviews stored separately, no aggregate ratings on parent entities
Likely Change: Add `average_rating` field to Program/Place/Trip models
Impact: Would need to calculate and cache average ratings, update on review creation

6.3 User-Specific Features

Current: Reviews linked to users, but no user-specific queries
Likely Change: Add "my reviews", "my favorites", user profiles
Impact: New endpoints like `GET /api/users/{user_id}/reviews`

6.4 Geographic Search

Current: Places have lat/long but not used for search
Likely Change: "Find places near me" within radius
Impact: Need geographic distance calculation in queries

6.5 Image Uploads

Current: Text-only descriptions
Likely Change: Support photo uploads for programs/places
Impact: Need file storage, image processing, new fields in models

6.6 Moderation System

Current: No review moderation
Likely Change: Admin approval for reviews, report inappropriate content
Impact: Add `status` field to reviews, admin endpoints

---

7. Testing Information

7.1 Test Coverage

The subsystem has been tested through:

1. **Manual API Testing** via Swagger UI at http://localhost:8000/docs
2. **Integration Tests** in `test_api.py`
3. **Frontend Integration Testing** - All endpoints called by React app

7.2 Running the Tests

Prerequisites:

```
 cd /path/to/Group14
# Ensure virtual environment is active and dependencies installed
uv sync
```

Start the Backend:

```
uv run uvicorn app.main:app --reload --port 8000
```

Run Automated Tests:

```
uv run python test_api.py
```

Interactive Testing (Swagger UI):

1. Navigate to http://localhost:8000/docs
2. Try each endpoint with sample data
3. All GET endpoints work without authentication
4. POST/PUT/DELETE require authentication (use `/auth/request-link` first)

7.3 Manual Test Scenarios

Test Case 1: Create and Retrieve Program

1. POST `/api/programs/` with program data (requires auth)
2. GET `/api/programs/` - should see new program
3. GET `/api/programs/{id}` - should return specific program

Test Case 2: Filter Programs

1. Create multiple programs in different cities
2. GET `/api/programs/?city=Paris` - should only return Paris programs
3. GET `/api/programs/?country=France` - should return all French programs

Test Case 3: Add Reviews

1. Create a program
2. POST `/api/programs/{id}/reviews` with rating and text
3. GET `/api/programs/{id}/reviews` - should return the review
4. Try invalid rating (0 or 6) - should fail validation

Test Case 4: Update Entity

1. Create a place
2. PUT `/api/places/{id}` with partial update (e.g., only description)
3. GET `/api/places/{id}` - should show updated description, other fields unchanged

Test Case 5: Delete with Reviews

1. Create a trip and add reviews
2. DELETE `/api/trips/{id}`
3. Verify trip and reviews are deleted (cascade)

7.4 Expected Behaviors

Status Codes:

- 200 OK: Successful GET, PUT
- 201 Created: Successful POST
- 204 No Content: Successful DELETE
- 401 Unauthorized: Missing auth on protected endpoints
- 404 Not Found: Entity doesn't exist
- 422 Unprocessable Entity: Validation errors (invalid rating, missing fields)

Authentication:

- All POST, PUT, DELETE require authentication
- All GET endpoints are public
- Authentication uses JWT cookies (handled by `current_user` dependency)

Validation:

- Ratings must be 1-5 (enforced at model level)
- Required fields must be provided on create
- Updates can be partial (only provided fields are updated)
- Foreign keys are validated (can't review non-existent program)

8. Review Focus Areas

We specifically request reviewers to focus on:

1. **Error Handling**: Are all edge cases covered? Should we add more validation?
2. **Security**: Are there any security vulnerabilities in the CRUD operations?
3. **Code Duplication**: The three API modules have very similar patterns - should we refactor?
4. **Performance**: Are the database queries efficient? Should we add eager loading for reviews?
5. **API Design**: Are the endpoints RESTful and intuitive? Any suggestions for improvement?
6. **Testing Gaps**: What additional tests would you recommend?

9. Development Team Notes

9.1 Tech Stack

- **Framework:** FastAPI (Python 3.11)
- **ORM:** SQLModel (SQLAlchemy + Pydantic)
- **Database:** SQLite (development), PostgreSQL (production planned)
- **Package Manager:** uv

9.2 Code Style

- Type hints on all function parameters and returns
- Docstrings on all public functions
- Follows REST conventions
- Ruff linting enabled (enforced in CI/CD)

9.3 Database Migrations

- Alembic for schema migrations
- Current migration: `81e660d3fa35_add_program_place_and_trip_models`
- All tables created with proper foreign keys and indexes

Appendix A: Quick Start Guide

Clone and Setup:

```
git clone https://github.com/slaterlucas/Group14.git
cd Group14
uv sync
```

Run Backend:

```
uv run uvicorn app.main:app --reload --port 8000
```

Access API Documentation:

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Seed Sample Data (Optional):

```
uv run python seed_data.py
```

Appendix B: Sample API Requests

Create Program:

```
POST http://localhost:8000/api/programs/
Content-Type: application/json

{
  "program_name": "Vanderbilt in Barcelona",
  "institution": "Vanderbilt University",
  "city": "Barcelona",
  "country": "Spain",
  "cost": 15000,
  "duration": "Summer (8 weeks)",
  "description": "Study Spanish culture and language"
}
```

List Places by Category:

```
GET http://localhost:8000/api/places/?category=restaurant&city=Paris
```

Add Trip Review:

```
POST http://localhost:8000/api/trips/5/reviews
Content-Type: application/json

{
  "rating": 5,
  "review_text": "Amazing weekend trip! Highly recommend the walking tour."
}
```

End of Documentation