CSCI 485: *Database Design & Collection Architecture*

Student ID: 664 870 797

Student name: Chris Lawrence

Project Title: EventSphere

A. Domain Analysis & Requirements Review

Recap of Domain and Primary Use Case

EventSphere is a MongoDB-backed events platform enabling users to discover, review, and attend in-person or virtual events. Core features include geospatial discovery near a location, full-text search, real-time updates, and analytics on attendance and reviews.

Key Queries to Support (5–8)

- Nearby events within X km of a location, optionally filtered by category and date window
- Full-text search across title/description/category/tags with relevance sorting
- Upcoming events (date-range filtering and sorting)
- Reviews by event and by venue, with rating aggregation
- Attendance analytics: repeat attendees, peak check-in hours, venue monthly stats
- User attendance history and event check-ins
- Category popularity and trends over time

Data Access Patterns & Performance Priorities

- Most frequent access: event discovery (geo + date + category) and text search
- Secondary: reviews retrieval and basic analytics aggregations
- Write patterns: event CRUD (moderate), check-ins (high volume bursts near event time), reviews (steady)
- Priorities:
    - Availability: browsing and search should remain responsive even under load
    - Consistency: strong consistency for booking/seat updates; eventual consistency acceptable for attendee counts, check-in feeds, and analytics

B. Collection Design Strategy

Collections Overview (minimum 4)

- `events`: Catalog of events with GeoJSON location, scheduling, embedded ticket tiers, and attendee snippets
- `venues`: Physical locations with address, capacity, amenities, and GeoJSON point
- `users`: User profiles and preferences (discovery filters, location, interests)
- `checkins`: Bridge collection for many-to-many user ↔ event attendance with analytics fields
- `reviews`: Feedback on events/venues with ratings and optional tags

Embedding vs Referencing Decisions

- Embedded
  - `events.tickets[]`: small, tightly bound to event; read together for listing/booking
  - `events.attendees[]` (lightweight snippet when used): quick RSVP display
  - `venues.address`: always co-read with venue
- Referenced
  - `events.venueId → venues._id`: venues shared by many events
  - `checkins.event_id`, `checkins.user_id`, `checkins.venue_id`: analytics-friendly fan-out
  - `reviews.event_id` or `reviews.venue_id` and `reviews.user_id`

Relationship Mapping & Justification

- 1:1 — venue:address (embedded subdocument for cohesion)
- 1:many — venue:events (reference from `events.venueId`) to avoid venue bloat
- many:many — users:events via `checkins` bridge to support analytics and scale; avoids unbounded arrays in `users` or `events`

C. Schema Design Documentation

Below, each collection includes: purpose/role, document structure, sample, validation highlights, and indexing strategy. Structures are aligned to `DATABASE_DESIGN.md` and generator scripts.

1) `events`

Purpose: Core catalog for discovery and analytics.

Document Structure (key fields):

- `_id: ObjectId`
- `title: String`
- `description: String`
- `category: String`
- `location: { type: "Point", coordinates: [lng:Number, lat:Number] }`
- `venueId: ObjectId | null`
- `start_date: Date`, `end_date: Date`
- `organizer: String`
- `max_attendees: Number`, `current_attendees: Number`
- `price: Number`, `currency: String`, `is_free: Boolean`
- `status: String` (draft|published|cancelled|completed)
- `tickets: [{ tier:String, price:Number, available:Number, sold:Number }]`
- `attendees: [{ user_id:ObjectId, checked_in:Boolean, check_in_time:Date }]`
- `tags: [String]`
- `metadata: { virtual:Boolean, recurring:Boolean, age_restriction:String, dress_code:String }`
- `created_at: Date`, `updated_at: Date`

Sample Document

```
      "title": String,        // Event title (indexed for text search) -
REQUIRED
      "description": String,  // Event description (indexed for text
search) - OPTIONAL
      "category": String,     // Event category (indexed) - REQUIRED
      "location": {           // GeoJSON Point for geospatial queries -
REQUIRED
        "type": "Point",    // Must be "Point"
        "coordinates": [longitude, latitude]
      },
      "venueId": ObjectId,
      "start_date": Date,
      "end_date": Date,
      "organizer": String,
      "max_attendees": Number,
      "current_attendees": Number,
      "price": Number,
      "currency": String,
      "is_free": Boolean,
      "status": String,
      "tickets": [{ "tier": String, "price": Number, "available": Number,
"sold": Number }],
      "attendees": [{ "user_id": ObjectId, "checked_in": Boolean,
"check_in_time": Date }],
      "tags": [String],
      "metadata": { "virtual": Boolean, "recurring": Boolean },
      "created_at": Date,
      "updated_at": Date
```

Validation Rules (highlights)

- Required: `title`, `category`, `location`, `start_date`, `created_at`, `updated_at`
- GeoJSON `location.type` enum `["Point"]`; `coordinates` length 2, numeric; bounds checks
- String length constraints; numeric minimums; `end_date` after `start_date`

Indexing Strategy

- `location: "2dsphere"` (geo)
- Text index on `title, description, category, tags`
- Single-field: `start_date`, `created_at`
- Compound: `{category:1,start_date:1}`, `{organizer:1,start_date:1}`, `{location:"2dsphere",start_date:1}`
- Pagination support: `{_id:1,start_date:1}`

2) `venues`

Purpose: Venue catalog for events and geo queries.

Key Fields

- `_id`, `name`, `type`, `description`

- `address { street, city, state, zip_code, country }`
- `location { type:"Point", coordinates:[lng,lat] }`
- `capacity: Number`, `amenities: [String]`, `contact { phone,email,website }`
- `pricing { hourly_rate, daily_rate, currency }`, `availability {...}`
- `rating: Number`, `review_count: Number`, `created_at`, `updated_at`

Sample Document

```
        "name": venue_name,
        "type": venue_type,
        "description": f"A {venue_type.lower()} located in {city['name']},
perfect for various events and gatherings.",
        "location": { "type": "Point", "coordinates": [lng, lat] },
        "address": { "street": "...", "city": city["name"], "state": "CA",
"zip_code": "...", "country": "USA" },
        "capacity": capacity,
        "amenities": ["WiFi", "Parking", ...],
        "contact": { "phone": "(555) 555-1234", "email": "info@...",
"website": "https://..." },
        "pricing": { "hourly_rate": 120, "daily_rate": 800, "currency":
"USD" },
        "availability": { "monday": {"open":"09:00","close":"22:00"}, ...
},
        "rating": 4.6,
        "review_count": 42,
        "created_at": ISODate(...),
        "updated_at": ISODate(...)
```

Validation & Indexes

- Require `name`, `address`, `location`, `created_at`
- Geo index: `location: "2dsphere"`
- Support text/filters via fields like `type`, `capacity`

3) `users`

Purpose: User profiles and discovery preferences.

Key Fields

- `_id`, `email`, `profile { first_name, last_name, preferences{ location, radius_km, categories[] } }`
- Additional app-profile fields for demo data (interests, bio, stats) may exist in generator outputs; core deliverable keeps minimal shape above
- `created_at`, `last_login`

Sample Document

```
  "_id": ObjectId,
  "email": String,
  "profile": {
    "first_name": String,
    "last_name": String,
    "preferences": {
      "categories": [String],
      "location": { "type": "Point", "coordinates": [lng, lat] },
      "radius_km": Number
    }
  },
  "created_at": Date,
  "last_login": Date
```

Validation & Indexes

- Require `email`, `profile`, `created_at`
- Optional Geo for preference location; unique email (optional for demo)

4) `checkins`

Purpose: Bridge collection for attendance (many:many) and analytics.

Key Fields

- `_id`, `event_id`, `user_id`, `venue_id`, `check_in_time`, `qr_code`, `ticket_tier`, `check_in_method`, `location`, `metadata{device_info, ip_address, staff_verified}`, `created_at`

Sample Document

```
  "_id": ObjectId,
  "event_id": ObjectId,
  "user_id": ObjectId,
  "venue_id": ObjectId,
  "check_in_time": Date,
  "qr_code": String,
  "ticket_tier": String,
  "check_in_method": String,
  "location": { "type": "Point", "coordinates": [lng, lat] },
  "metadata": { "device_info": String, "ip_address": String,
"staff_verified": Boolean },
  "created_at": Date
```

Validation & Indexes

- Require all referenced ids, `check_in_time`, `qr_code`, `created_at`
- Unique pair index to prevent duplicates: `{event_id:1, user_id:1}`

- Indexes for analytics: `event_id`, `user_id`, `venue_id`, `check_in_time`, `qr_code`, plus compound as needed

5) `reviews`

Purpose: Ratings/comments on events or venues.

Key Fields

- `_id`, `event_id?`, `venue_id?`, `user_id`, `rating(1–5)`, `comment`, `created_at`, `updated_at`

Sample Document

```
"_id": ObjectId,
"event_id": ObjectId,
"venue_id": ObjectId,
"user_id": ObjectId,
"rating": Number,
"comment": String,
"created_at": Date,
"updated_at": Date
```

Validation & Indexes

- Require `user_id`, `rating`, `created_at`, `updated_at` and exactly one of `event_id` or `venue_id`
- Indexes: `event_id`, `venue_id`, `user_id`, `rating`, `created_at`, and compounds for aggregations

D. Implementation Artifacts (Scripts)

See accompanying files under `docs/viu/deliverables/2/`:

- `create_collections.js` — Creates collections with `$jsonSchema` validators
- `sample_data.js` — Inserts realistic sample documents showing relationships
- `create_indexes.js` — Adds initial strategic indexes with comments

These align with playgrounds (`01_crud.mongodb`, `02_text_search.mongodb`, `03_geo.mongodb`) and the indexing plan from `DATABASE_DESIGN.md`.