Low-Level Design (LLD) - Mini-Trello Kanban App

Database Schema Design

1. Users Collection
\`\`\`javascript
{
  _id: ObjectId,
  name: String(required, max: 100),
  email: String(required, unique, lowercase),
  password: String(required, hashed),
  avatar: String(URL),
  workspaces: [ObjectId(ref: Workspace)],
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.users.createIndex({ email: 1 }, { unique: true })
db.users.createIndex({ workspaces: 1 })
\`\`\`

2. Workspaces Collection
\`\`\`javascript
{
  _id: ObjectId,
  name: String(required, max: 100),
  description: String(max: 500),
  owner: ObjectId(ref: User, required),
  members: [{
    user: ObjectId(ref: User, required),
    role: String(enum: ['admin', 'member'], default: 'member'),
    joinedAt: Date(default: now)
  }],
  boards: [ObjectId(ref: Board)],
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.workspaces.createIndex({ owner: 1 })
db.workspaces.createIndex({ "members.user": 1 })

db.workspaces.createIndex({ owner: 1, "members.user": 1 })
\`\`\`

3. Boards Collection
\`\`\`javascript
{
  _id: ObjectId,
  title: String(required, max: 100),
  description: String(max: 500),
  workspace: ObjectId(ref: Workspace, required),
  owner: ObjectId(ref: User, required),
  visibility: String(enum: ['private', 'workspace', 'public'], default: 'workspace'),
  members: [{
    user: ObjectId(ref: User, required),
    role: String(enum: ['owner', 'admin', 'member', 'viewer'], default: 'member'),
    joinedAt: Date(default: now)
  }],
  lists: [ObjectId(ref: List)],
  labels: [{
    name: String(required, max: 50),
    color: String(required, regex: /^#[0-9A-F]{6}$/i)
  }],
  background: String(default: '#0079bf'),
  starred: [ObjectId(ref: User)],
  closed: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.boards.createIndex({ workspace: 1 })
db.boards.createIndex({ owner: 1 })
db.boards.createIndex({ "members.user": 1 })
db.boards.createIndex({ visibility: 1 })
db.boards.createIndex({ workspace: 1, closed: 1 })
\`\`\`

4. Lists Collection
\`\`\`javascript
{
  _id: ObjectId,
  title: String(required, max: 100),
  board: ObjectId(ref: Board, required),
  position: Number(required, default: 1024),

```
  cards: [ObjectId(ref: Card)],
  archived: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.lists.createIndex({ board: 1, position: 1 })
db.lists.createIndex({ board: 1, archived: 1 })
```

5. Cards Collection
```javascript
{
  _id: ObjectId,
  title: String(required, max: 200),
  description: String(max: 2000),
  list: ObjectId(ref: List, required),
  board: ObjectId(ref: Board, required),
  position: Number(required, default: 1024),
  labels: [ObjectId], // References to board.labels
  assignees: [ObjectId(ref: User)],
  dueDate: Date,
  completed: Boolean(default: false),
  completedAt: Date,
  creator: ObjectId(ref: User, required),
  attachments: [{
    name: String(required),
    url: String(required),
    size: Number,
    uploadedBy: ObjectId(ref: User, required),
    uploadedAt: Date(default: now)
  }],
  checklist: [{
    text: String(required, max: 200),
    completed: Boolean(default: false),
    completedAt: Date,
    completedBy: ObjectId(ref: User)
  }],
  comments: [ObjectId(ref: Comment)],
  archived: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}
```

// Indexes
db.cards.createIndex({ list: 1, position: 1 })
db.cards.createIndex({ board: 1, archived: 1 })
db.cards.createIndex({ assignees: 1 })
db.cards.createIndex({ dueDate: 1 })
db.cards.createIndex({ title: "text", description: "text" })
db.cards.createIndex({ board: 1, list: 1, position: 1 })
\`\`\`

6. Comments Collection
\`\`\`javascript
{
  _id: ObjectId,
  text: String(required, max: 1000),
  card: ObjectId(ref: Card, required),
  author: ObjectId(ref: User, required),
  edited: Boolean(default: false),
  editedAt: Date,
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.comments.createIndex({ card: 1, createdAt: -1 })
db.comments.createIndex({ author: 1, createdAt: -1 })
\`\`\`

7. Activities Collection
\`\`\`javascript
{
  _id: ObjectId,
  type: String(required, enum: [
    'card_created', 'card_updated', 'card_moved', 'card_archived', 'card_deleted',
    'comment_added', 'comment_updated', 'comment_deleted',
    'list_created', 'list_updated', 'list_moved', 'list_archived',
    'board_created', 'board_updated',
    'member_added', 'member_removed',
    'label_added', 'label_removed',
    'due_date_added', 'due_date_updated', 'due_date_removed',
    'attachment_added', 'attachment_removed',
    'checklist_item_added', 'checklist_item_completed', 'checklist_item_uncompleted'
  ]),
  actor: ObjectId(ref: User, required),

```
  board: ObjectId(ref: Board, required),
  card: ObjectId(ref: Card),
  list: ObjectId(ref: List),
  comment: ObjectId(ref: Comment),
  data: Mixed, // Additional context data
  description: String(required, max: 500),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.activities.createIndex({ board: 1, createdAt: -1 })
db.activities.createIndex({ card: 1, createdAt: -1 })
db.activities.createIndex({ actor: 1, createdAt: -1 })
db.activities.createIndex({ type: 1, createdAt: -1 })
\`\`\`

## API Definitions

### Authentication Endpoints

#### POST /api/auth/register
\`\`\`javascript
// Request
{
  name: "John Doe",
  email: "john@example.com",
  password: "securepassword123"
}

// Response (201)
{
  token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  user: {
    id: "507f1f77bcf86cd799439011",
    name: "John Doe",
    email: "john@example.com",
    avatar: null
  }
}

// Error Response (400)
{
  message: "User already exists",
```

```
  errors: [
    {
      field: "email",
      message: "Email is already registered"
    }
  ]
}
\`\`\`

#### POST /api/auth/login
\`\`\`javascript
// Request
{
  email: "john@example.com",
  password: "securepassword123"
}

// Response (200)
{
  token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  user: {
    id: "507f1f77bcf86cd799439011",
    name: "John Doe",
    email: "john@example.com",
    avatar: "https://example.com/avatar.jpg"
  }
}
\`\`\`

### Board Management Endpoints

#### GET /api/boards
\`\`\`javascript
// Headers
Authorization: Bearer <token>

// Response (200)
[
  {
    _id: "507f1f77bcf86cd799439011",
    title: "Website Redesign",
    description: "Complete website overhaul project",
    workspace: {
      _id: "507f1f77bcf86cd799439012",
```

```
      name: "Acme Corp"
    },
    background: "#0079bf",
    visibility: "workspace",
    starred: false,
    updatedAt: "2024-01-15T10:30:00Z"
  }
]
\`\`\`

#### POST /api/boards
\`\`\`javascript
// Request
{
  title: "New Project Board",
  description: "Project description",
  workspaceId: "507f1f77bcf86cd799439012",
  visibility: "workspace",
  background: "#519839"
}

// Response (201)
{
  _id: "507f1f77bcf86cd799439013",
  title: "New Project Board",
  description: "Project description",
  workspace: "507f1f77bcf86cd799439012",
  owner: "507f1f77bcf86cd799439011",
  visibility: "workspace",
  background: "#519839",
  members: [
    {
      user: {
        _id: "507f1f77bcf86cd799439011",
        name: "John Doe",
        email: "john@example.com"
      },
      role: "owner",
      joinedAt: "2024-01-15T10:30:00Z"
    }
  ],
  lists: [],
  labels: [],
  createdAt: "2024-01-15T10:30:00Z"
```

}
\`\`\`

GET /api/boards/:id
\`\`\`javascript
// Response (200) - Full board with lists and cards
{
  _id: "507f1f77bcf86cd799439013",
  title: "Website Redesign",
  workspace: {
    _id: "507f1f77bcf86cd799439012",
    name: "Acme Corp"
  },
  members: [...],
  lists: [
    {
      _id: "507f1f77bcf86cd799439014",
      title: "To Do",
      position: 1024,
      cards: [
        {
          _id: "507f1f77bcf86cd799439015",
          title: "Design homepage mockup",
          description: "Create initial design concepts",
          position: 1024,
          assignees: [...],
          labels: [...],
          dueDate: "2024-01-20T00:00:00Z",
          comments: [...],
          checklist: [...]
        }
      ]
    }
  ],
  labels: [
    {
      _id: "507f1f77bcf86cd799439016",
      name: "High Priority",
      color: "#eb5a46"
    }
  ]
}
\`\`\`

Card Management Endpoints

POST /api/cards
\`\`\`javascript
// Request
{
  title: "Implement user authentication",
  description: "Add JWT-based authentication system",
  listId: "507f1f77bcf86cd799439014",
  position: 2048
}

// Response (201)
{
  _id: "507f1f77bcf86cd799439017",
  title: "Implement user authentication",
  description: "Add JWT-based authentication system",
  list: "507f1f77bcf86cd799439014",
  board: "507f1f77bcf86cd799439013",
  position: 2048,
  creator: {
    _id: "507f1f77bcf86cd799439011",
    name: "John Doe"
  },
  assignees: [],
  labels: [],
  comments: [],
  checklist: [],
  attachments: [],
  createdAt: "2024-01-15T10:30:00Z"
}
\`\`\`

PUT /api/cards/:id/move
\`\`\`javascript
// Request
{
  listId: "507f1f77bcf86cd799439018", // New list ID
  position: 1536 // New position
}

// Response (200)
{
  _id: "507f1f77bcf86cd799439017",

```
  list: "507f1f77bcf86cd799439018",
  position: 1536,
  // ... other card fields
}
```

Real-time Socket Events

Client → Server Events
```javascript
// Join board room
socket.emit('join-board', boardId);

// Leave board room
socket.emit('leave-board', boardId);

// Card moved
socket.emit('card-moved', {
  cardId: '507f1f77bcf86cd799439017',
  fromListId: '507f1f77bcf86cd799439014',
  toListId: '507f1f77bcf86cd799439018',
  newPosition: 1536,
  boardId: '507f1f77bcf86cd799439013'
});

// Typing indicators
socket.emit('typing-start', {
  cardId: '507f1f77bcf86cd799439017',
  boardId: '507f1f77bcf86cd799439013'
});
```

Server → Client Events
```javascript
// User joined board
socket.on('user-joined-board', (data) => {
  // data: { userId, userName, socketId }
});

// Card moved by another user
socket.on('card-moved', (data) => {
  // data: { cardId, fromListId, toListId, newPosition, userId, userName }
});
```

```
// New comment added
socket.on('comment-added', (data) => {
  // data: { comment, cardId, boardId, userId, userName }
});
\`\`\`

## Position-based Ordering Strategy

### Fractional Positioning Algorithm
\`\`\`javascript
// Calculate position for new item
function calculatePosition(prevItem, nextItem) {
  if (!prevItem && !nextItem) {
    return 1024; // First item
  }

  if (!prevItem) {
    return nextItem.position / 2; // Insert at beginning
  }

  if (!nextItem) {
    return prevItem.position + 1024; // Insert at end
  }

  // Insert between items
  return (prevItem.position + nextItem.position) / 2;
}

// Rebalance positions when they get too close
function rebalancePositions(items) {
  if (items.length === 0) return;

  const step = 1024;
  items.forEach((item, index) => {
    item.position = (index + 1) * step;
  });
}

// Check if rebalancing is needed
function needsRebalancing(items) {
  for (let i = 1; i < items.length; i++) {
    if (items[i].position - items[i-1].position < 1) {
      return true;
    }
```

```
  }
  return false;
}
\`\`\`

### Position Update Flow
\`\`\`javascript
// Client-side optimistic update
const moveCard = async (cardId, newListId, newPosition) => {
  // 1. Update UI immediately
  updateCardPositionInUI(cardId, newListId, newPosition);

  // 2. Emit real-time event
  socket.emit('card-moved', { cardId, newListId, newPosition });

  // 3. Send API request
  try {
    await api.put(`/cards/${cardId}/move`, {
      listId: newListId,
      position: newPosition
    });
  } catch (error) {
    // 4. Revert on error
    revertCardPosition(cardId);
    showError('Failed to move card');
  }
};
\`\`\`

## Error Handling Model

### Error Response Format
\`\`\`javascript
// Standard error response
{
  success: false,
  message: "Human-readable error message",
  code: "ERROR_CODE",
  errors: [
    {
      field: "fieldName",
      message: "Field-specific error message",
      code: "FIELD_ERROR_CODE"
    }
```

```
  ],
  timestamp: "2024-01-15T10:30:00Z",
  requestId: "req_123456789"
}
```

Error Categories
```javascript
// Authentication Errors (401)
const AUTH_ERRORS = {
  INVALID_TOKEN: 'Token is invalid or expired',
  MISSING_TOKEN: 'Authorization token is required',
  INVALID_CREDENTIALS: 'Invalid email or password'
};

// Authorization Errors (403)
const AUTHZ_ERRORS = {
  ACCESS_DENIED: 'You do not have permission to access this resource',
  BOARD_ACCESS_DENIED: 'You are not a member of this board',
  WORKSPACE_ACCESS_DENIED: 'You are not a member of this workspace'
};

// Validation Errors (400)
const VALIDATION_ERRORS = {
  REQUIRED_FIELD: 'This field is required',
  INVALID_FORMAT: 'Invalid format for this field',
  DUPLICATE_VALUE: 'This value already exists'
};

// Resource Errors (404)
const RESOURCE_ERRORS = {
  BOARD_NOT_FOUND: 'Board not found',
  CARD_NOT_FOUND: 'Card not found',
  USER_NOT_FOUND: 'User not found'
};
```

Error Handling Middleware
```javascript
// Global error handler
const errorHandler = (err, req, res, next) => {
  const error = {
    success: false,
    message: err.message || 'Internal server error',
```

```
    timestamp: new Date().toISOString(),
    requestId: req.id
  };

  // Log error details
  logger.error({
    error: err,
    request: {
      method: req.method,
      url: req.url,
      user: req.user?.id,
      body: req.body
    }
  });

  // Handle specific error types
  if (err.name === 'ValidationError') {
    error.code = 'VALIDATION_ERROR';
    error.errors = Object.values(err.errors).map(e => ({
      field: e.path,
      message: e.message,
      code: 'INVALID_VALUE'
    }));
    return res.status(400).json(error);
  }

  if (err.name === 'JsonWebTokenError') {
    error.code = 'INVALID_TOKEN';
    error.message = 'Invalid authentication token';
    return res.status(401).json(error);
  }

  // Default server error
  error.code = 'INTERNAL_ERROR';
  error.message = process.env.NODE_ENV === 'production'
    ? 'Internal server error'
    : err.message;

  res.status(err.status || 500).json(error);
};
\`\`\`

## Concurrency & Conflict Resolution
```

Last-Write-Wins Strategy
\`\`\`javascript
// Card update with version checking
const updateCard = async (cardId, updates, version) => {
  const card = await Card.findById(cardId);

  if (!card) {
    throw new Error('Card not found');
  }

  // Simple version check (using updatedAt)
  if (version && card.updatedAt.getTime() !== version) {
    throw new ConflictError('Card has been modified by another user');
  }

  // Apply updates
  Object.assign(card, updates);
  card.updatedAt = new Date();

  await card.save();

  // Broadcast update
  io.to(`board-${card.board}`).emit('card-updated', {
    cardId,
    updates,
    version: card.updatedAt.getTime()
  });

  return card;
};
\`\`\`

Position Conflict Resolution
\`\`\`javascript
// Handle position conflicts during card moves
const resolvePositionConflict = async (listId, targetPosition) => {
  const cards = await Card.find({ list: listId })
    .sort({ position: 1 });

  // Check for position conflicts
  const conflictingCard = cards.find(card =>
    Math.abs(card.position - targetPosition) < 0.1
  );

```
  if (conflictingCard) {
    // Rebalance all positions in the list
    const step = 1024;
    for (let i = 0; i < cards.length; i++) {
      cards[i].position = (i + 1) * step;
      await cards[i].save();
    }

    // Return new safe position
    return cards.length * step + step;
  }

  return targetPosition;
};
\`\`\`
```

This low-level design provides the detailed implementation specifications needed to build a robust, scalable Mini-Trello application with proper error handling, conflict resolution, and performance optimization strategies.

Low-Level Design (LLD) - Mini-Trello Kanban App

Database Schema Design

1. Users Collection
```
\`\`\`javascript
{
  _id: ObjectId,
  name: String(required, max: 100),
  email: String(required, unique, lowercase),
  password: String(required, hashed),
  avatar: String(URL),
  workspaces: [ObjectId(ref: Workspace)],
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.users.createIndex({ email: 1 }, { unique: true })
db.users.createIndex({ workspaces: 1 })
\`\`\`
```

2. Workspaces Collection
\`\`\`javascript
{
  _id: ObjectId,
  name: String(required, max: 100),
  description: String(max: 500),
  owner: ObjectId(ref: User, required),
  members: [{
    user: ObjectId(ref: User, required),
    role: String(enum: ['admin', 'member'], default: 'member'),
    joinedAt: Date(default: now)
  }],
  boards: [ObjectId(ref: Board)],
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.workspaces.createIndex({ owner: 1 })
db.workspaces.createIndex({ "members.user": 1 })
db.workspaces.createIndex({ owner: 1, "members.user": 1 })
\`\`\`

3. Boards Collection
\`\`\`javascript
{
  _id: ObjectId,
  title: String(required, max: 100),
  description: String(max: 500),
  workspace: ObjectId(ref: Workspace, required),
  owner: ObjectId(ref: User, required),
  visibility: String(enum: ['private', 'workspace', 'public'], default: 'workspace'),
  members: [{
    user: ObjectId(ref: User, required),
    role: String(enum: ['owner', 'admin', 'member', 'viewer'], default: 'member'),
    joinedAt: Date(default: now)
  }],
  lists: [ObjectId(ref: List)],
  labels: [{
    name: String(required, max: 50),
    color: String(required, regex: /^#[0-9A-F]{6}$/i)
  }],
  background: String(default: '#0079bf'),
  starred: [ObjectId(ref: User)],

```
  closed: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.boards.createIndex({ workspace: 1 })
db.boards.createIndex({ owner: 1 })
db.boards.createIndex({ "members.user": 1 })
db.boards.createIndex({ visibility: 1 })
db.boards.createIndex({ workspace: 1, closed: 1 })
```

4. Lists Collection
```javascript
{
  _id: ObjectId,
  title: String(required, max: 100),
  board: ObjectId(ref: Board, required),
  position: Number(required, default: 1024),
  cards: [ObjectId(ref: Card)],
  archived: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.lists.createIndex({ board: 1, position: 1 })
db.lists.createIndex({ board: 1, archived: 1 })
```

5. Cards Collection
```javascript
{
  _id: ObjectId,
  title: String(required, max: 200),
  description: String(max: 2000),
  list: ObjectId(ref: List, required),
  board: ObjectId(ref: Board, required),
  position: Number(required, default: 1024),
  labels: [ObjectId], // References to board.labels
  assignees: [ObjectId(ref: User)],
  dueDate: Date,
  completed: Boolean(default: false),
```

```
  completedAt: Date,
  creator: ObjectId(ref: User, required),
  attachments: [{
    name: String(required),
    url: String(required),
    size: Number,
    uploadedBy: ObjectId(ref: User, required),
    uploadedAt: Date(default: now)
  }],
  checklist: [{
    text: String(required, max: 200),
    completed: Boolean(default: false),
    completedAt: Date,
    completedBy: ObjectId(ref: User)
  }],
  comments: [ObjectId(ref: Comment)],
  archived: Boolean(default: false),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.cards.createIndex({ list: 1, position: 1 })
db.cards.createIndex({ board: 1, archived: 1 })
db.cards.createIndex({ assignees: 1 })
db.cards.createIndex({ dueDate: 1 })
db.cards.createIndex({ title: "text", description: "text" })
db.cards.createIndex({ board: 1, list: 1, position: 1 })
\`\`\`

### 6. Comments Collection
\`\`\`javascript
{
  _id: ObjectId,
  text: String(required, max: 1000),
  card: ObjectId(ref: Card, required),
  author: ObjectId(ref: User, required),
  edited: Boolean(default: false),
  editedAt: Date,
  createdAt: Date,
  updatedAt: Date
}

// Indexes
```

db.comments.createIndex({ card: 1, createdAt: -1 })
db.comments.createIndex({ author: 1, createdAt: -1 })
\`\`\`

7. Activities Collection
\`\`\`javascript
{
  _id: ObjectId,
  type: String(required, enum: [
    'card_created', 'card_updated', 'card_moved', 'card_archived', 'card_deleted',
    'comment_added', 'comment_updated', 'comment_deleted',
    'list_created', 'list_updated', 'list_moved', 'list_archived',
    'board_created', 'board_updated',
    'member_added', 'member_removed',
    'label_added', 'label_removed',
    'due_date_added', 'due_date_updated', 'due_date_removed',
    'attachment_added', 'attachment_removed',
    'checklist_item_added', 'checklist_item_completed', 'checklist_item_uncompleted'
  ]),
  actor: ObjectId(ref: User, required),
  board: ObjectId(ref: Board, required),
  card: ObjectId(ref: Card),
  list: ObjectId(ref: List),
  comment: ObjectId(ref: Comment),
  data: Mixed, // Additional context data
  description: String(required, max: 500),
  createdAt: Date,
  updatedAt: Date
}

// Indexes
db.activities.createIndex({ board: 1, createdAt: -1 })
db.activities.createIndex({ card: 1, createdAt: -1 })
db.activities.createIndex({ actor: 1, createdAt: -1 })
db.activities.createIndex({ type: 1, createdAt: -1 })
\`\`\`

API Definitions

Authentication Endpoints

POST /api/auth/register
\`\`\`javascript
// Request

```
{
  name: "John Doe",
  email: "john@example.com",
  password: "securepassword123"
}

// Response (201)
{
  token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  user: {
    id: "507f1f77bcf86cd799439011",
    name: "John Doe",
    email: "john@example.com",
    avatar: null
  }
}

// Error Response (400)
{
  message: "User already exists",
  errors: [
    {
      field: "email",
      message: "Email is already registered"
    }
  ]
}
\`\`\`

#### POST /api/auth/login
\`\`\`javascript
// Request
{
  email: "john@example.com",
  password: "securepassword123"
}

// Response (200)
{
  token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  user: {
    id: "507f1f77bcf86cd799439011",
    name: "John Doe",
    email: "john@example.com",
```

```
    avatar: "https://example.com/avatar.jpg"
  }
}
\`\`\`

### Board Management Endpoints

#### GET /api/boards
\`\`\`javascript
// Headers
Authorization: Bearer <token>

// Response (200)
[
  {
    _id: "507f1f77bcf86cd799439011",
    title: "Website Redesign",
    description: "Complete website overhaul project",
    workspace: {
      _id: "507f1f77bcf86cd799439012",
      name: "Acme Corp"
    },
    background: "#0079bf",
    visibility: "workspace",
    starred: false,
    updatedAt: "2024-01-15T10:30:00Z"
  }
]
\`\`\`

#### POST /api/boards
\`\`\`javascript
// Request
{
  title: "New Project Board",
  description: "Project description",
  workspaceId: "507f1f77bcf86cd799439012",
  visibility: "workspace",
  background: "#519839"
}

// Response (201)
{
  _id: "507f1f77bcf86cd799439013",
```

```
  title: "New Project Board",
  description: "Project description",
  workspace: "507f1f77bcf86cd799439012",
  owner: "507f1f77bcf86cd799439011",
  visibility: "workspace",
  background: "#519839",
  members: [
    {
      user: {
        _id: "507f1f77bcf86cd799439011",
        name: "John Doe",
        email: "john@example.com"
      },
      role: "owner",
      joinedAt: "2024-01-15T10:30:00Z"
    }
  ],
  lists: [],
  labels: [],
  createdAt: "2024-01-15T10:30:00Z"
}
\`\`\`

#### GET /api/boards/:id
\`\`\`javascript
// Response (200) - Full board with lists and cards
{
  _id: "507f1f77bcf86cd799439013",
  title: "Website Redesign",
  workspace: {
    _id: "507f1f77bcf86cd799439012",
    name: "Acme Corp"
  },
  members: [...],
  lists: [
    {
      _id: "507f1f77bcf86cd799439014",
      title: "To Do",
      position: 1024,
      cards: [
        {
          _id: "507f1f77bcf86cd799439015",
          title: "Design homepage mockup",
          description: "Create initial design concepts",
```

```
          position: 1024,
          assignees: [...],
          labels: [...],
          dueDate: "2024-01-20T00:00:00Z",
          comments: [...],
          checklist: [...]
        }
      ]
    }
  ],
  labels: [
    {
      _id: "507f1f77bcf86cd799439016",
      name: "High Priority",
      color: "#eb5a46"
    }
  ]
}
\`\`\`

### Card Management Endpoints

#### POST /api/cards
\`\`\`javascript
// Request
{
  title: "Implement user authentication",
  description: "Add JWT-based authentication system",
  listId: "507f1f77bcf86cd799439014",
  position: 2048
}

// Response (201)
{
  _id: "507f1f77bcf86cd799439017",
  title: "Implement user authentication",
  description: "Add JWT-based authentication system",
  list: "507f1f77bcf86cd799439014",
  board: "507f1f77bcf86cd799439013",
  position: 2048,
  creator: {
    _id: "507f1f77bcf86cd799439011",
    name: "John Doe"
  },
```

```
  assignees: [],
  labels: [],
  comments: [],
  checklist: [],
  attachments: [],
  createdAt: "2024-01-15T10:30:00Z"
}
\`\`\`

#### PUT /api/cards/:id/move
\`\`\`javascript
// Request
{
  listId: "507f1f77bcf86cd799439018", // New list ID
  position: 1536 // New position
}

// Response (200)
{
  _id: "507f1f77bcf86cd799439017",
  list: "507f1f77bcf86cd799439018",
  position: 1536,
  // ... other card fields
}
\`\`\`

### Real-time Socket Events

#### Client → Server Events
\`\`\`javascript
// Join board room
socket.emit('join-board', boardId);

// Leave board room
socket.emit('leave-board', boardId);

// Card moved
socket.emit('card-moved', {
  cardId: '507f1f77bcf86cd799439017',
  fromListId: '507f1f77bcf86cd799439014',
  toListId: '507f1f77bcf86cd799439018',
  newPosition: 1536,
  boardId: '507f1f77bcf86cd799439013'
});
```

```
// Typing indicators
socket.emit('typing-start', {
  cardId: '507f1f77bcf86cd799439017',
  boardId: '507f1f77bcf86cd799439013'
});
\`\`\`

#### Server → Client Events
\`\`\`javascript
// User joined board
socket.on('user-joined-board', (data) => {
  // data: { userId, userName, socketId }
});

// Card moved by another user
socket.on('card-moved', (data) => {
  // data: { cardId, fromListId, toListId, newPosition, userId, userName }
});

// New comment added
socket.on('comment-added', (data) => {
  // data: { comment, cardId, boardId, userId, userName }
});
\`\`\`

## Position-based Ordering Strategy

### Fractional Positioning Algorithm
\`\`\`javascript
// Calculate position for new item
function calculatePosition(prevItem, nextItem) {
  if (!prevItem && !nextItem) {
    return 1024; // First item
  }

  if (!prevItem) {
    return nextItem.position / 2; // Insert at beginning
  }

  if (!nextItem) {
    return prevItem.position + 1024; // Insert at end
  }
```

```
  // Insert between items
  return (prevItem.position + nextItem.position) / 2;
}

// Rebalance positions when they get too close
function rebalancePositions(items) {
  if (items.length === 0) return;

  const step = 1024;
  items.forEach((item, index) => {
    item.position = (index + 1) * step;
  });
}

// Check if rebalancing is needed
function needsRebalancing(items) {
  for (let i = 1; i < items.length; i++) {
    if (items[i].position - items[i-1].position < 1) {
      return true;
    }
  }
  return false;
}
\`\`\`

### Position Update Flow
\`\`\`javascript
// Client-side optimistic update
const moveCard = async (cardId, newListId, newPosition) => {
  // 1. Update UI immediately
  updateCardPositionInUI(cardId, newListId, newPosition);

  // 2. Emit real-time event
  socket.emit('card-moved', { cardId, newListId, newPosition });

  // 3. Send API request
  try {
    await api.put(`/cards/${cardId}/move`, {
      listId: newListId,
      position: newPosition
    });
  } catch (error) {
    // 4. Revert on error
    revertCardPosition(cardId);
```

```
    showError('Failed to move card');
  }
};
\`\`\`

## Error Handling Model

### Error Response Format
\`\`\`javascript
// Standard error response
{
  success: false,
  message: "Human-readable error message",
  code: "ERROR_CODE",
  errors: [
    {
      field: "fieldName",
      message: "Field-specific error message",
      code: "FIELD_ERROR_CODE"
    }
  ],
  timestamp: "2024-01-15T10:30:00Z",
  requestId: "req_123456789"
}
\`\`\`

### Error Categories
\`\`\`javascript
// Authentication Errors (401)
const AUTH_ERRORS = {
  INVALID_TOKEN: 'Token is invalid or expired',
  MISSING_TOKEN: 'Authorization token is required',
  INVALID_CREDENTIALS: 'Invalid email or password'
};

// Authorization Errors (403)
const AUTHZ_ERRORS = {
  ACCESS_DENIED: 'You do not have permission to access this resource',
  BOARD_ACCESS_DENIED: 'You are not a member of this board',
  WORKSPACE_ACCESS_DENIED: 'You are not a member of this workspace'
};

// Validation Errors (400)
const VALIDATION_ERRORS = {
```

```
  REQUIRED_FIELD: 'This field is required',
  INVALID_FORMAT: 'Invalid format for this field',
  DUPLICATE_VALUE: 'This value already exists'
};

// Resource Errors (404)
const RESOURCE_ERRORS = {
  BOARD_NOT_FOUND: 'Board not found',
  CARD_NOT_FOUND: 'Card not found',
  USER_NOT_FOUND: 'User not found'
};
\`\`\`

### Error Handling Middleware
\`\`\`javascript
// Global error handler
const errorHandler = (err, req, res, next) => {
  const error = {
    success: false,
    message: err.message || 'Internal server error',
    timestamp: new Date().toISOString(),
    requestId: req.id
  };

  // Log error details
  logger.error({
    error: err,
    request: {
      method: req.method,
      url: req.url,
      user: req.user?.id,
      body: req.body
    }
  });

  // Handle specific error types
  if (err.name === 'ValidationError') {
    error.code = 'VALIDATION_ERROR';
    error.errors = Object.values(err.errors).map(e => ({
      field: e.path,
      message: e.message,
      code: 'INVALID_VALUE'
    }));
    return res.status(400).json(error);
```

```
  }

  if (err.name === 'JsonWebTokenError') {
    error.code = 'INVALID_TOKEN';
    error.message = 'Invalid authentication token';
    return res.status(401).json(error);
  }

  // Default server error
  error.code = 'INTERNAL_ERROR';
  error.message = process.env.NODE_ENV === 'production'
    ? 'Internal server error'
    : err.message;

  res.status(err.status || 500).json(error);
};
\`\`\`

## Concurrency & Conflict Resolution

### Last-Write-Wins Strategy
\`\`\`javascript
// Card update with version checking
const updateCard = async (cardId, updates, version) => {
  const card = await Card.findById(cardId);

  if (!card) {
    throw new Error('Card not found');
  }

  // Simple version check (using updatedAt)
  if (version && card.updatedAt.getTime() !== version) {
    throw new ConflictError('Card has been modified by another user');
  }

  // Apply updates
  Object.assign(card, updates);
  card.updatedAt = new Date();

  await card.save();

  // Broadcast update
  io.to(`board-${card.board}`).emit('card-updated', {
    cardId,
```

```
    updates,
    version: card.updatedAt.getTime()
  });

  return card;
};
```

Position Conflict Resolution
```javascript
// Handle position conflicts during card moves
const resolvePositionConflict = async (listId, targetPosition) => {
  const cards = await Card.find({ list: listId })
    .sort({ position: 1 });

  // Check for position conflicts
  const conflictingCard = cards.find(card =>
    Math.abs(card.position - targetPosition) < 0.1
  );

  if (conflictingCard) {
    // Rebalance all positions in the list
    const step = 1024;
    for (let i = 0; i < cards.length; i++) {
      cards[i].position = (i + 1) * step;
      await cards[i].save();
    }

    // Return new safe position
    return cards.length * step + step;
  }

  return targetPosition;
};
```

This low-level design provides the detailed implementation specifications needed to build a robust, scalable Mini-Trello application with proper error handling, conflict resolution, and performance optimization strategies.