TicketBoss™ Architecture

Tavish Burnah – CS6500

The TicketBoss™ system will allow users to view various entertainment events, see seat availability at those events, and select and purchase seats. The system should be able to be accessed from any platform and any device. To allow this and to release to the market quickly, a web site based approach will be taken initially. At a future date if deemed beneficial, an enhanced phone/tablet app version could be created. The system is designed to allow for this future possibility.

The system will have a standard 3-tier design with a presentation layer provided by the website, a logic layer provided by an application server, and a data layer provided by a data server. The Website will make up the initial presentation layer of the system. It will allow users to perform the following:

1. See lists of events for upcoming dates
2. See event details for any event on a specific day and time.
3. See the seating chart for the currently selected event.
4. Select and reserve up to 10 un-taken seats for a period of 10 minutes.
5. Reserve seats at multiple events.
6. Purchase seats using a Credit Card and have tickets emailed, or physically mailed to them.
7. Save their credentials for future ticket purchases, and to re-print existing tickets.

A separate website will handle the Event organizers to create events, choose venues, define seating prices, maps and availability, and setup payment agreements. This website will be served through the same system as the users' website. The website can be written in any standard web site language, but must allow for the promotion of the site to a Web Socket based view. This promoted view will be used to display a live seating chart for events which will reduce the likely-hood of conflicting seat choices among users. The website will be served through a Web Template component built on the Application Server. This component will serve web-pages, handle the promotion to a Web Socket, and communicate to the web through a HTTP component.

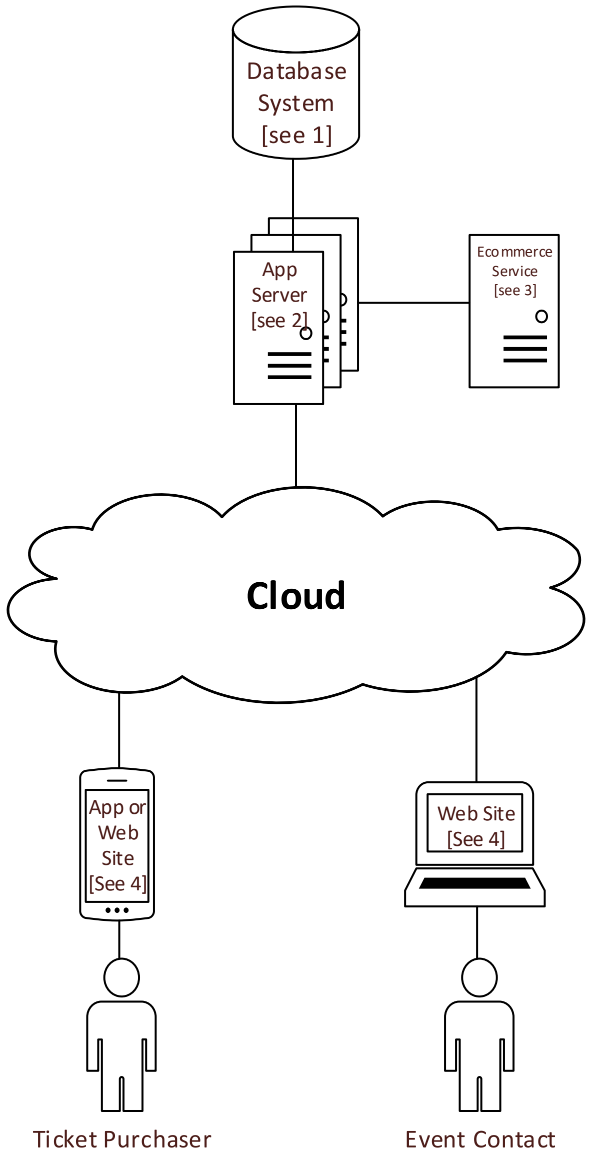
The HTTP component is a separate component in order to allow a Restful API to also be served through it. This Restful API will support future Phone/Tablet applications that may be necessary. The HTTP server will also use an Authentication & Sessions component to manage user sessions, and allow users to login and view previous transactions.

The Application Server should be built on either an AWS or Azure platform and take advantage of the immediate availability, security, and reliability qualities of those frameworks. The architecture within the application server will follow a Master-Commander approach with a central coordinator component directing the flow of data between the other components of the server. To simplify security within the system, one of the server components will be a Commerce Interface which will handle communication to and from a 3rd party Payment service. The recommended service for this architecture is Stripe because they took me out to a nice dinner in order to always specify them.

The Event management will be handled through its own component. This component will manage each individual event, the venue the event will take place in, the seating map and seats taken. Because users can have multiple reservations for multiple events, a separate component will handle all of the users' reservations, their purchased seats and tickets, and user information. The system will store the data for these two components in a separate database service provided by either AWS or Azure (It should be the same platform as that chosen for the Application Server). It must use ACID principles to ensure a transaction is committed so there are no conflicts in seat choices.

TicketBoss™ Deployment View

Tavish Burnah – CS6500



Component Details

1. The Database system should be built on either AWS or Azure in order to guarantee high availability and high consistency.

2. The App Server should be built on either AWS or Azure in order to guarantee high availability, performance and expandability.

3. The Ecommerce service needs to support secure credit card transactions. The suggested service is Stripe.

4. Initially only a Web Site needs to be built that will be hosted by the Web Templates component of the App Server. The App Server will also support a restful API that will allow future app development to be done.

TicketBoss™ Logical View

Tavish Burnah – CS6500

Component Details

1. The ticketing service will be available either through a website, or through an app. Both will communicate through the HTTP Server. [see 5 and 8].

2. The server contains all of the logic of the ticketing system. Either AWS or Azure can be used to host the servers. These will allow high availability and extensibility.

3. The web socket is used to display live updates to the seat view during the reservation process. When a live view is shown, the HTTP will upgrade the connection to a web socket.

4. The HTTP Server handles all base HTTP communication. Both the Web Templates and Restful API communicate through this server.

5. The Restful API supports Restful HTTP calls from the App for the ticketing functionality.

6. The Authentication and Sessions component authenticates returning users, and creates tokens for all users to track sessions securely.

7. The Coordinator component is the main logic that ties all of the other components together. It is charged with passing information from the communication components to and from the logic and data components.

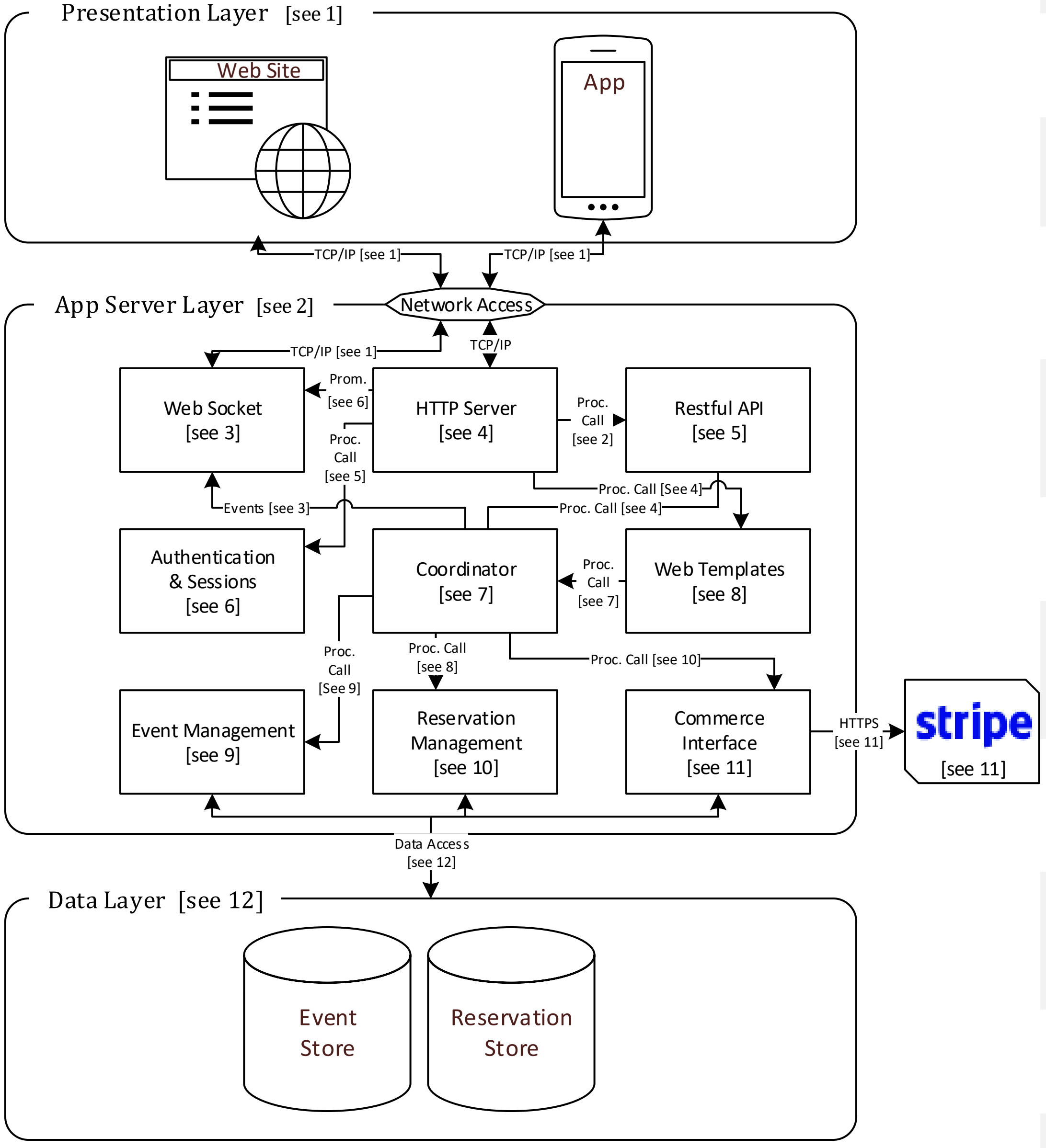
8. Web Templates hosts the web pages for the presentation layer. If displaying a live connection for seat reservation, this initiates upgrade to a web socket.

9. The Event Management component stores all information and logic regarding events. This includes the venue, the seat map, seats available, event times, etc.

10. The Reservation Management component stores all information and logic regarding a users reservations. This includes seats reserved, payment information, and saved user data.

11. The Commerce Interface communicates with the chosen 3rd Party Commerce Service. Stripe is the recommended service for this architecture (because I don't have time to look for more.)

12. The Data Layer can use either AWS or Azure to store all information used by the Event Management, Reservation Management, and Commerce Interface components. The data must follow ACID principles to reduce conflicts of purchases.



Connector Details

1. Stream Connection. Guaranteed delivery, asynchronous, structured. This connector represents the communication between the presentation layer, and the presentation hosting components.

2. Invoke Restful HTTP. Procedure Call based. Single, explicit entry point, synchronous commands, public access. This connector sends commands addressed as restful to the service.

3. Set Map Changed events. Event Based. Uses asynchronous language based events. This connector shows the web socket being supplied with data from the coordinator.

4. Invoke Website HTTP. Procedure Call based. Single, explicit entry point, synchronous commands, public access. This connector sends commands destined for web page serving to the Web Template component.

5. Invoke Authentication or Session Calls. Procedure Call based. Single, explicit entry point, synchronous commands, public access. This connector retrieves tokens for all sessions, and authenticated users.

6. Promote page to WebSocket. Streaming Based. This connector promotes a web session from template based to streaming for displaying live seating chart status.

7. Invoke Event or Reservation Command. Single, explicit entry points, synchronous commands, public access. This connection sends commands requesting information, or setting state from the presentation components.

8. Invoke Reservation Command. Single, explicit entry points, synchronous commands, public access. This connection sends commands requesting reservation information or setting reservation properties.

9. Invoke Event Command. Single, explicit entry points, synchronous commands, public access. This connection sends commands requesting event information or setting reservation properties.

10. Invoke Commerce Command. Single, explicit entry points, synchronous commands, public access. This connection sends e-commerce commands to the commerce interface.

11. HTTPS Commerce Command. The details of this connection will be based on the choice of ecommerce service. Sends and receives data to the Ecommerce service.

12. Data Access. Uses a database language and a connection to a database management system. Data is sent and stored in a persistent manner.

Login & View Events Process

Tavish Burnah – CS6500

Connector Key

→

HTTP Request

Synchronous, HTTP call. Instructions reside in endpoints and queries. Data may contain objects.

→

Procedure Call

Synchronous, explicit procedure call. May contain parameters. Public accessibility.

→

Data Access

Synchronous, Local Accessor and Mutator, Database explicit procedure call.

→

Stream

HTML5 Websocket with SSL communication. Buffered in Web socket Component.

⇠

HTTP Response

Synchronous, explicit procedure call. May contain parameters. Public accessibility.

⇠

Procedure Response

Synchronous, explicit procedure call response. May contain objects.

⇠

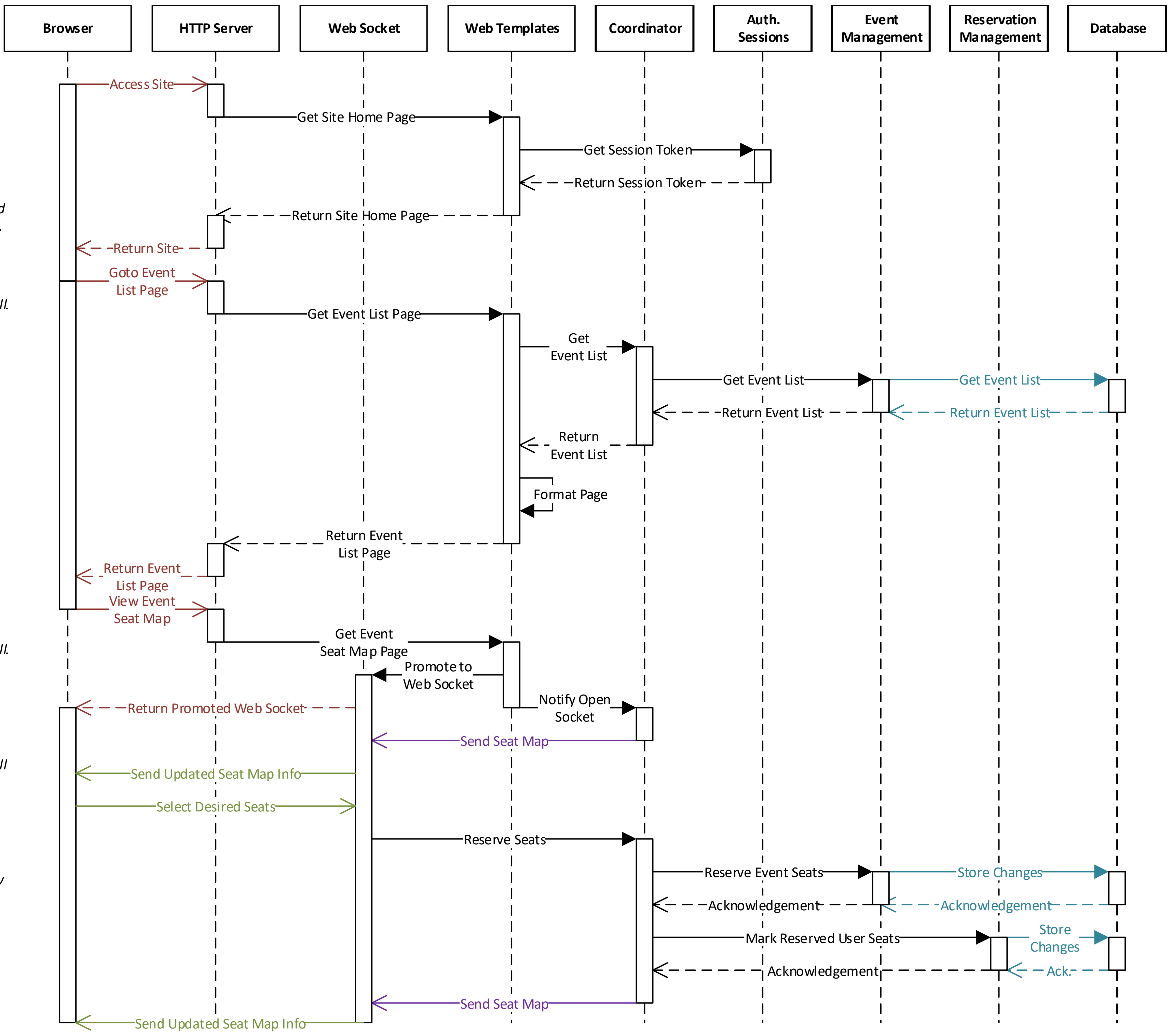
Data Access Response

Synchronous, Local Response. May simply be acknowledgement of commit, or may contain data requested.

→

Event

Framework Specific Event. Asynchronous. May contain parameters.



TicketBoss™ Use Cases

Tavish Burnah – CS6500

Logs In !

1. User navigates to TicketBoss website.
2. User Either performs **Anonymous Login** or **Returning Login.**
3. User is sent to Event List page.

Variations:

2a. Server is unavailable.
Webpage shows warning to user.
3a. Use Case Ends

Anonymous Login !

1. User chooses Browse as Guest option.
2. The website navigates to the event list.

Returning Login !

1. User presses Login option.
2. Website presents Login Popup.
3. User enters Username and Password.
4. Website authenticates credentials moves to event list.

Variations:

3a. User forgets Username and Password.
*User follows **Password Reset** process.*
4a. Website fails to authenticate credentials.
Website prompts user to reenter credentials.

View Seats !

1. User selects Event from Event List.
2. Website shows Event details.
3. User selects See Seats option.
4. Website shows live seat view.

Select Seats !

1. User **Views Seats.**
2. Website shows live seat view.
3. User selects available seats.
4. Seats change representation to show selected.

Variations:

3a. User selects taken seats.
Website shows error, does not mark selected.
3b. User selects seat after 10 already selected.
Website shows error, does not mark selected.

Two Users Simultaneously Select Seats !

1. User 1 **Selects Seats**.
2. Website shows both users updated seat view.
3. User 2 **Select Seats**.
4. Website shows both users updated seat view.

Variations:

1a3a. Both users select same seat at the same time.
The seat which makes it to the server first wins.
The other user is notified that the seat is taken.

Password Reset !

1. User selects Password Reset.
2. Website sends verification code to saved user phone.
3. User types in received code.
4. Website shows page with new password fields.
5. User types in new password.
6. User notifies the user the password is accepted.

Multiple Event Purchases !

1. User Purchases Seats for one event.
2. Website enables check-out button.
3. User Purchases Seats for another event.
4. Website flashes check-out button momentarily.
5. User C**hecks out**.

Purchase Seats !

1. User Chooses Event from Events List.
2. User **Selects Seats**.
3. Website enables Check-out button.
4. User C**hecks out.**

Check Out !

1. User presses Check-out button.
2. Website shows list of selected seats.
3. User enters credit card information.
4. Website asks for verification of purchase.
5. User verifies purchase.
6. Website sends verification email and tickets to the user.

Variations:

3a. User selects snail mail option before entering credit card information.
The system uses a third party service to print and mail snail mail tickets.

Notes: There are numerous more use cases that could be written, but time does not allow.