Service Architecture Review vs Rubric Requirements

11/23/2025

Solution Review

01

OPTIONS FOR SERVICE DELIVERY

02

SERVICE ARCHITECTURE DESIGN CHOICES

03

SERVICE ARCHITECTURE VS RUBRIC

OPTIONS FOR SERVICE DELIVERY

SECTION 01

Solution Review

01

OPTIONS FOR SERVICE DELIVERY

02

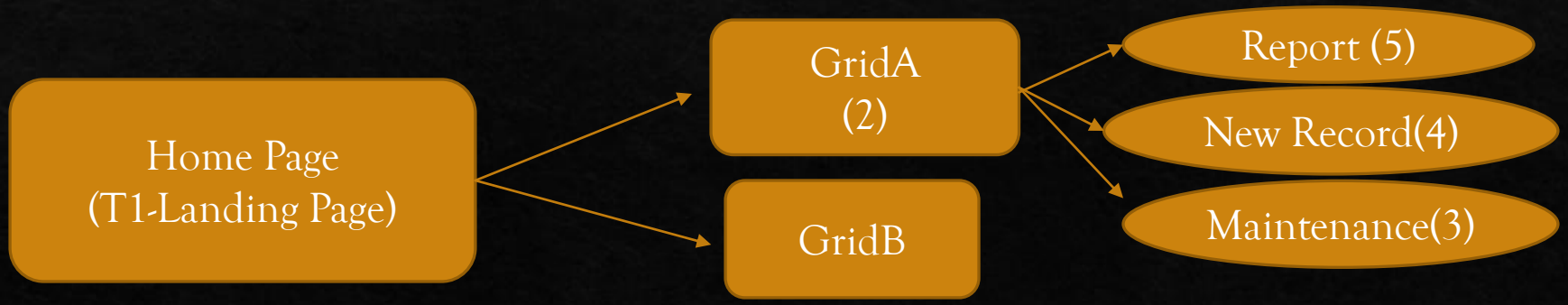
SERVICE ARCHITECTURE DESIGN CHOICES

03

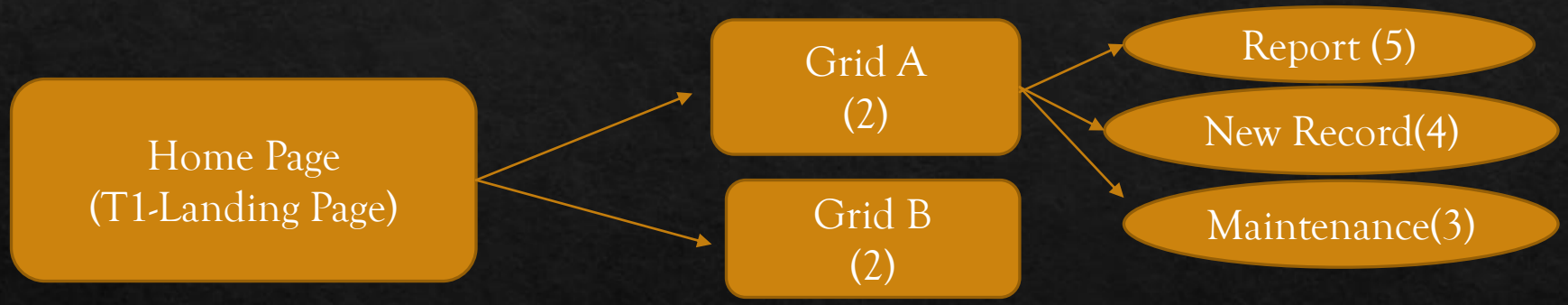
SERVICE ARCHITECTURE VS RUBRIC

Services vs Normal Database Operations(1)

- In Software Engineering we usually build screens as Units of Work.
  - Screens Usually Are Of Five Main Types
    - 1) Landing Pages (Home, HR, OPS), Welcome/Dashoards, Etc
    - 2) Selection Grids of Work – Getting All Existing Records So that One or More Can Be Selected. (Get/Select)
    - 3) Maintenance Operations For Existing Work After Selection (Mutation of Existing Records – Update/Change, Delete)
    - 4) Managing Forms as New Units of Work (New Records).
    - 5) Reports which usually are presented to a screen before Redirection to a Printer
  - Regardless of the Interface Type We Usually Have A Grid as the Top Level



Services vs Normal Database Operations(2)



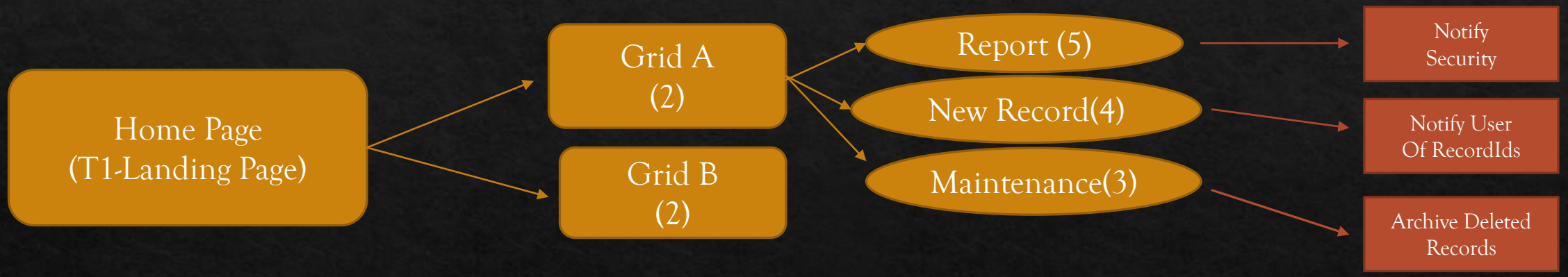
When Performing Work on Child Record Pages we can:

a) Simply Maintain and Add Data to A Single Table, or a set of Relations.
b) Maintain & Add Records But TAKING OTHER ACTIONS

ACTIONS can be:

a) Updating Other Tables with Overlapping Data.
b) Alerting the Customer to their Web GUIs or Mobile Clients.
c) Emailing the Customer Details of Transactions.

Services vs Normal Database Operations(3)



1) Home Pages Almost Never Generate Actions And Therefore very Infrequently require service logic EXCEPT for FirstUse logic which may need to run an installer on your Mobile Device, Workstation, etc.

2) Grid Selection Pages almost never require service logic except for the acknowledgement of alarms.

3) Reports usually just display data, they don't usually take other actions except in high security systems.

4) New Record Creation(Type4) and Maintenance Activities(Type3) however commonly have service logic as well as database operations behind them.

547/590 Targets

Service Architectures Options

- UI TO Gmail (Fall2024-Team1)
- UI -> MVC -> C# SERVICE (TARGET)
- UI->MVC->SQL->SQL TRIGGER

590 Service Architectures Required

- UI -> REST->AZURE CLOUD SVCS
- UI-> MVC->SQL->SQL TRIGGER

Overall Service Options More Expansive

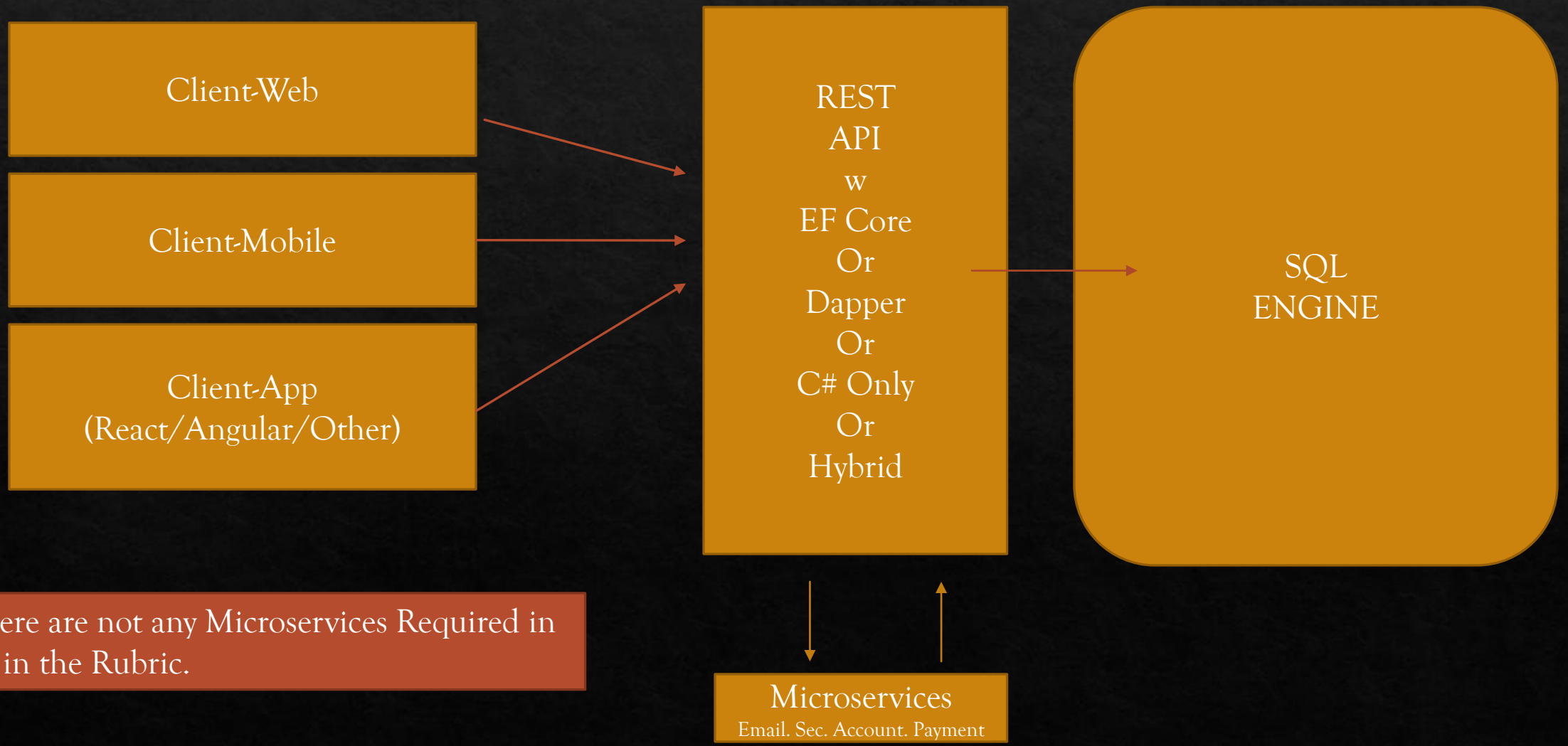
Service Types

- Services Architectures Can Vary Dramatically
  - UI Services run all logic in a Thick Client.
  - UI only with IOT fulfillment
  - UI with Server Side Actions(7 Types)
  - UI with REST MVC
  - UI with AJAX OTHER
  - SQL Server Driven Services(Triggers, Email)
  - UI to Cloud
  - Hybrid (Combination of UI Driven, IoT, Cloud, MVC)

Classifications

- Suggested Classifications
  - UI-Only
  - UI with IoT Enhancement
  - UIwSSP(PHP), UIwSSL(LINQ/ASPX/RAZOR), UIwSSCGI(CGI), UIwSSCFM, U
  - RESTMVC
  - AJAXOTHER
  - SQLACTIONS
  - UICLOUD
  - HYBRID

Visual Studio Model View Controllers



There are not any Microservices Required in S1 in the Rubric.

Rubric Requirements

- Services for 547 Are Not Required until Sprint2:
  - Rubric Requires A Reservation to Update Park Inventory
  - Rubric Requires A Reservation to Email User the Booking(Extra Credit)

- Note, Common User, Auth and Security Services are not in the Rubric.

Sprints For Dirtbike Project

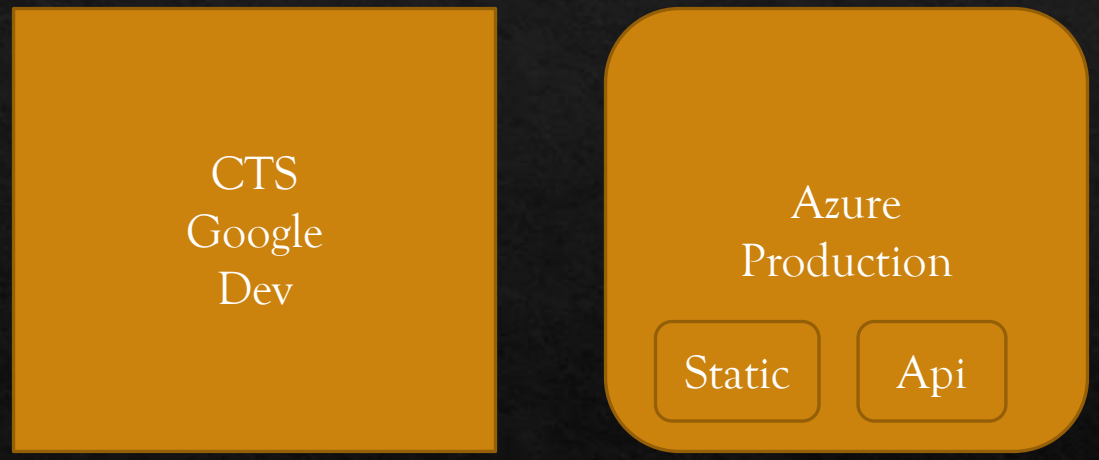
- Sprint0-> Reasonable Project Plan and Architectural Decisions.
- Sprint1-> Demonstration of Backoffice Capabilities
- Sprint2-> End to End Delivery->Service Fulfilmment.

Team5 Service Delivery

Core Tasks for Sprint1

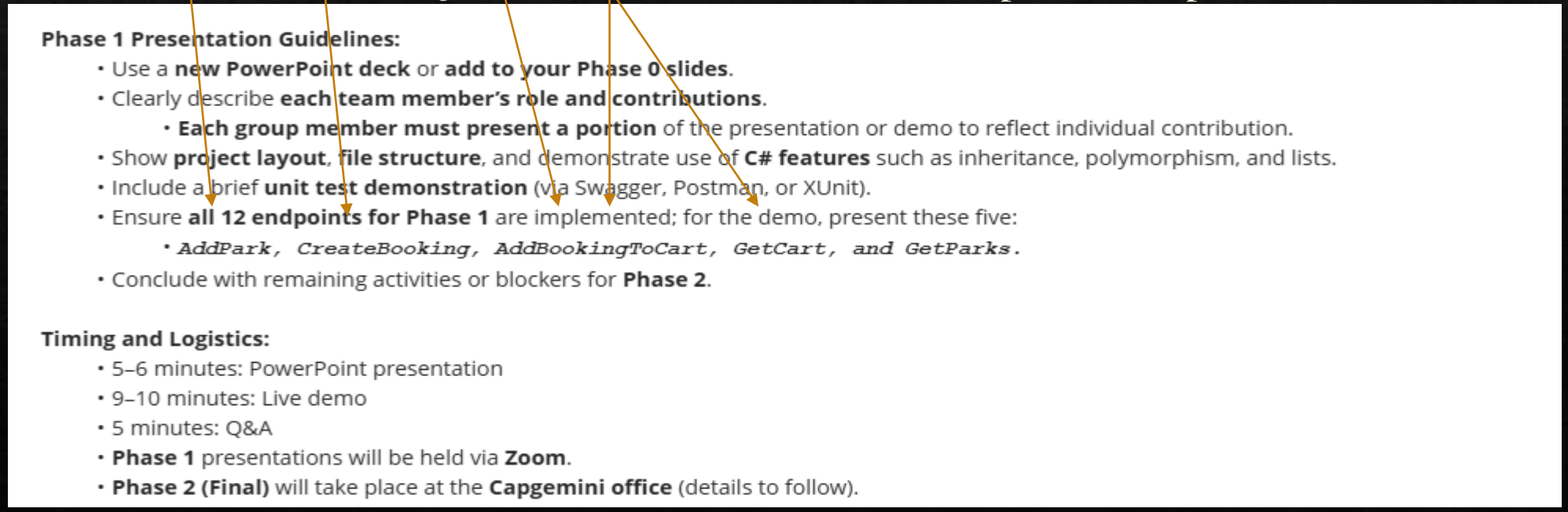
We Explained... Our Model had morphed

- We started out planning to do hosting on two Platforms (CTS/Azure) using the same app.
- However we found that its hard to do this during development phases. It quadruples operational steps.
- We moved to using CTS/Google Cloud VMs for Development
- Azure Cloud using Static Apps, and APIs like 590 is the Production Environment.
- We rebuilt the User interfaces/Auth so that we could run on both Google, and Azure without the 590 Azure SQL Repositories.



S1 Rubric

- Requires Park(1), Booking(2), Cart(3), Payment APIs (4) x CRUD or 16 Endpoints to be complete while the Rubric Lists 12. [We Disagree that Add Booking to Cart is an EP, As We Agreed that Bookings are After Payment - Kyle].
- At a minimum four(4) SQL tables are needed to accomplish this process.



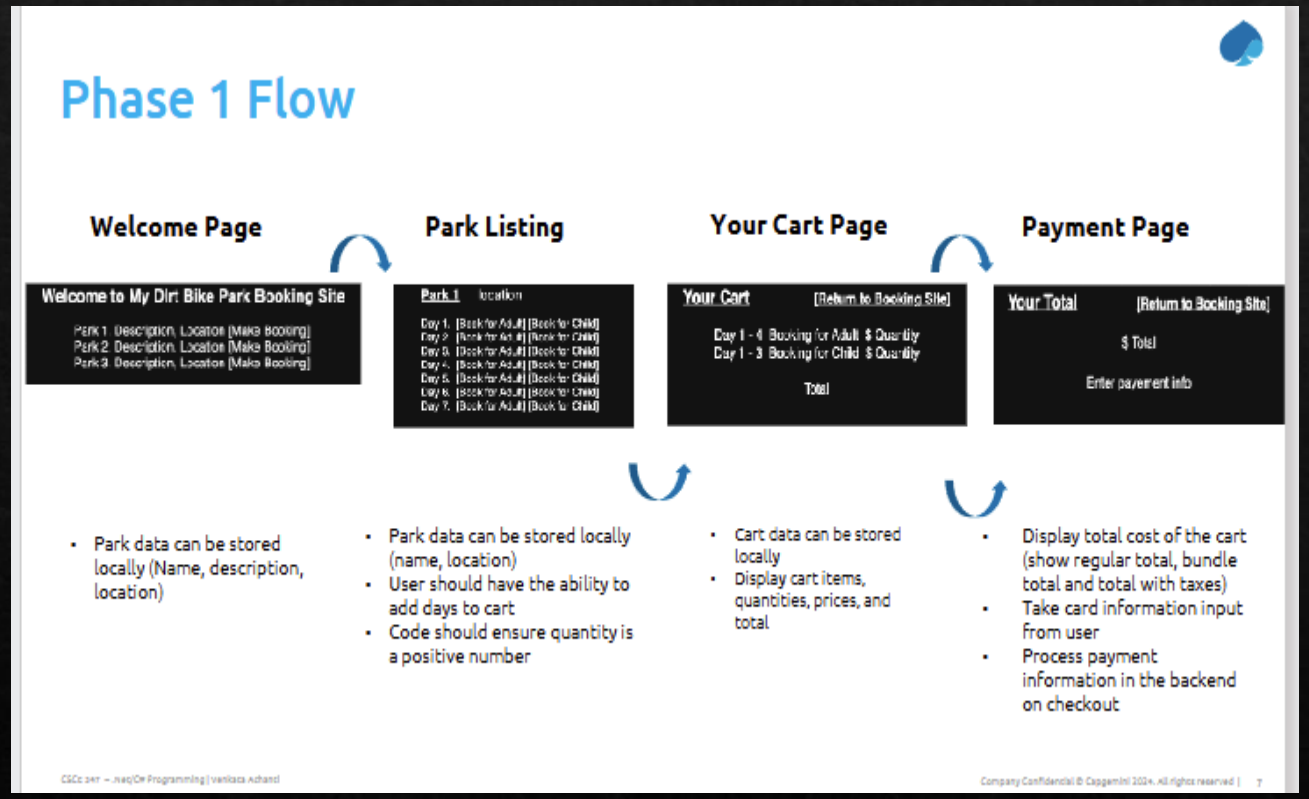
Phase 1 Presentation Guidelines:

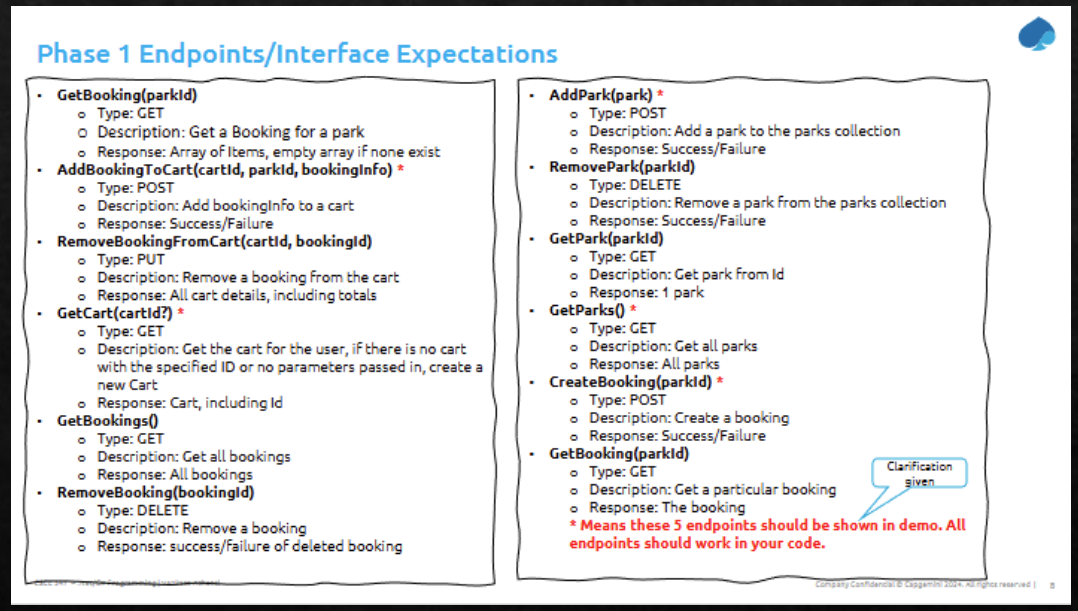
- Use a **new PowerPoint deck** or **add to your Phase 0 slides**.
- Clearly describe **each team member's role and contributions**.
  - **Each group member must present a portion** of the presentation or demo to reflect individual contribution.
- Show **project layout**, **file structure**, and demonstrate use of **C# features** such as inheritance, polymorphism, and lists.
- Include a brief **unit test demonstration** (via Swagger, Postman, or XUnit).
- Ensure **all 12 endpoints for Phase 1** are implemented; for the demo, present these five:
  - ***AddPark, CreateBooking, AddBookingToCart, GetCart, and GetParks.***
- Conclude with remaining activities or blockers for **Phase 2**.

Timing and Logistics:

- 5–6 minutes: PowerPoint presentation
- 9–10 minutes: Live demo
- 5 minutes: Q&A
- **Phase 1** presentations will be held via **Zoom**.
- **Phase 2 (Final)** will take place at the **Capgemini office** (details to follow).

Rubric 12 Master for S1 vs S1Preso





S1 Requirements Vs Demo....

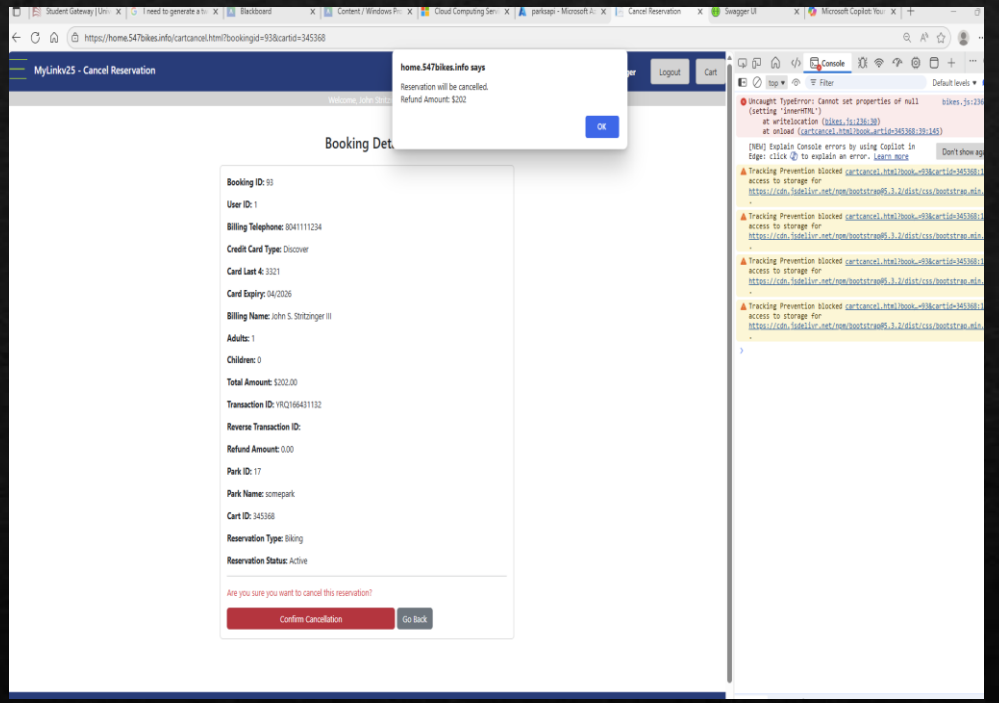
- **BEFORE SPRINT 1 – WE PROVIDED A LIVE DEMO SITE TO OUR WORKING APPLICATION. ALL ISSUES WITH TECHNICAL DESIGN COULD BE RESOLVED VIA THE UI PROVIDED.**
- S1 Required End to End Flow Demonstration
  - We Presented an Order from WWW->PARKS->CART->PAYMENT->BOOKING
  - We presented the ability to Choose a Park from 150 Parks+, Create a Booking, and Follow that through the Booking/Reservation Grid.
  - We demonstrated the Reservation Grid, and Showed we could remove a reservation and credit back cards.
  - We built a Custom UI (Which wasn't required), and More than 30 Screens with full S2 Requirements for CRUD.
  - We Showed AUTH Capabilities in our UI using Team590 User Architectures.
  - We demonstrated the EndState Security Portal with Working Security Services.

T5 vs Other Teams

Testing

Testing Model

- Phase 1
  - We used swagger alone initially to do single table operations. 1 Controller. 1 Endpoint.
- Phase 2
  - We used swagger with Console.log to get UI running.
- Phase 3
  - We used End to End Full Transactions after UI, and Backoffice were working.



We have run 106 Live Booking Transactions… just on production, and at least 2X that on Development Server. More than 300 Unit Tests X 6 steps.

T1,T2,T3

- Don’t have a working UI.
- They have not run an end to end transaction with payments at all.
- Most if not all failed their S1 demos or couldn’t get swagger to work during the live presentation.

Solution Review

01

OPTIONS FOR SERVICE DELIVERY

02

SERVICE ARCHITECTURE DESIGN CHOICES

03

SERVICE ARCHITECTURE VS RUBRIC

S1 Required EP Collection #1

Parks

General SQL Information

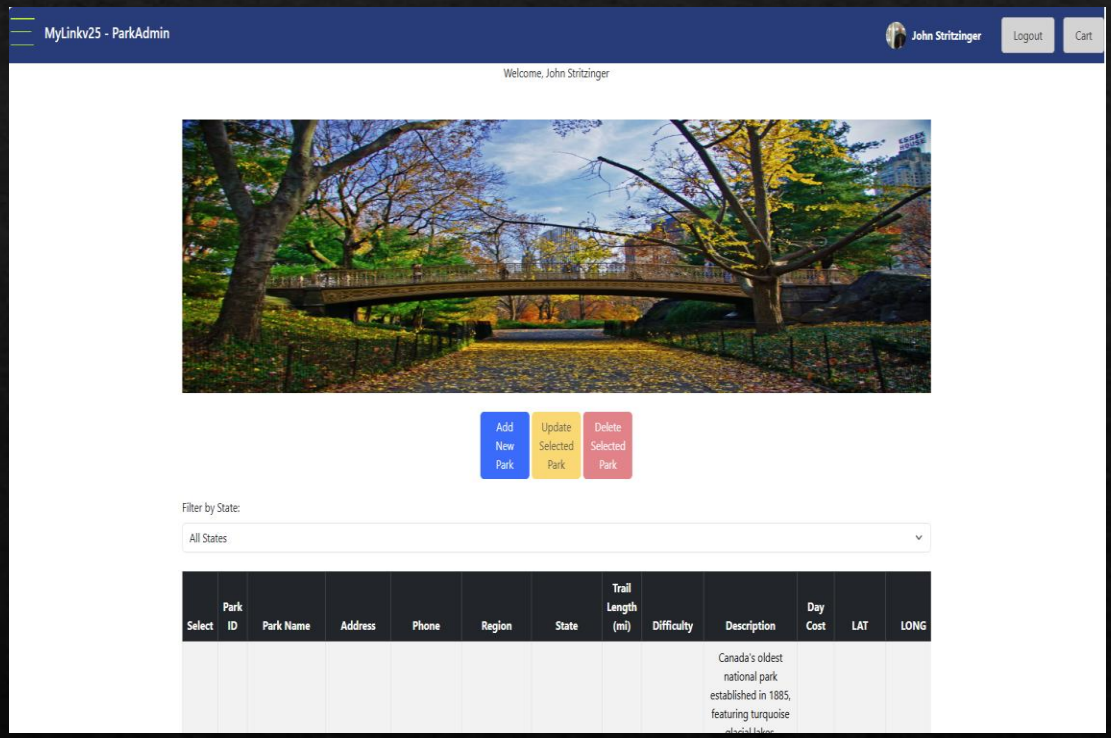
- SQLITE supports all standard SQL Operations.
- SQLITE supports Views, and Instantiated Views.
- SQLITE supports Triggers, and Automation post Insert/Update.
- SQLITE supports autoincrement counters even for Deleted Records. Next record is not reset to n-1.
- Very impressed overall with its capabilities.

For All Demonstrated Endpoints

- We have 1 Table for Each Endpoint we are using.
- EG Parks has 1 SQL table driving 6+ Endpoints ON SQLITE.
- All are built from JSS's Templates Designed by Team1 from 590. (Ghost Squad).
- We are using a 1 C# file in Visual Studio for Each TableName.
- Parks has Parks.cs both as a Model in the Models Directory and as a Controller in the Controllers Directory.
- Endpoints without modification and speciality requests in our template are 4 per table by Autoincremented Ids. UserID driven takes this to 6EPs per table.

S1->Maintaining Parks

- Team5 Demonstrated the Park Maintenance Screen. It Presents A Grid of Existing Parks, and Supports Selection and Updating of Data of any of the 150+ Parks.
- We also demonstrated in Excess of the Rubric the Park Data being demonstrated in Park Visualizations
- We also demonstrated How we got the Model data using AI tools to our JSON schema.



S1-Park Supporting Requirements (API)
parksapi.547bikes.info/swagger

PARKS ->SQL LITE DEFINITION

Keys

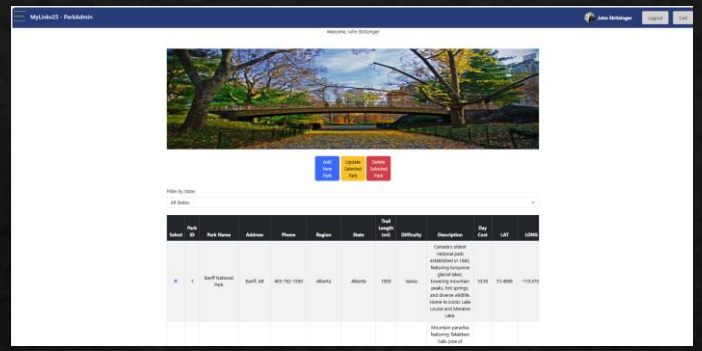
parkID – identity, autogenerated, unique

No Triggers, Or Constraints for S1

```
CREATE TABLE Parks (
    parkId INTEGER PRIMARY KEY, name TEXT, address TEXT, phone TEXT, region TEXT, trailLengthMiles REAL, difficulty TEXT, description TEXT, dayPassPriceUsd REAL,
longitude REAL, latitude REAL, trailmapurl TEXT, parklogourl TEXT,
    maxvisitors INTEGER, currentvisitors INTEGER, currentvisitorschildren INTEGER, currentvisitorsadults INTEGER, maxcampsites INTEGER, columns INTEGER, state TEXT, pic1url
TEXT, pic2url TEXT, pic3url TEXT, pic4url TEXT, pic5url TEXT,
    pic6url TEXT, pic7url TEXT, pic8url TEXT, pic9url TEXT, isnationalpark TEXT, isstatepark TEXT, hqbranchid TEXT, mountainbikes INTEGER, camping INTEGER, rafting
INTEGER, canoeing INTEGER, frisbee INTEGER, iscanadian INTEGER,
    ismexican INTEGER, motocross INTEGER, cabins INTEGER, tents INTEGER, skiing INTEGER
);
```

Significant Upgrades to Parks were done to support Driving Directions, and Other Park Services on the same REST endpoints.

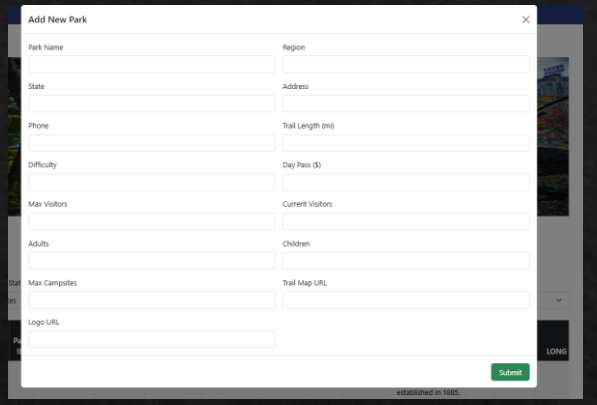
Park Architecture



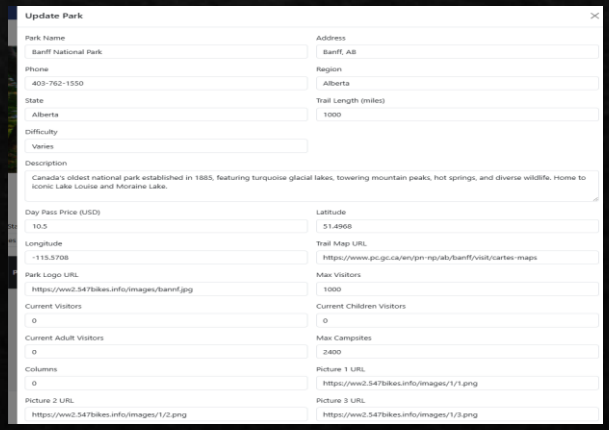
Grid Architecture Using Bootstrap Layout with a Get Call to /api/Parks

Modal Add

Modal Update



Adding a New Park Requires a POST EP in Working Form.



Updating an Existing Park Requires a PUT EP

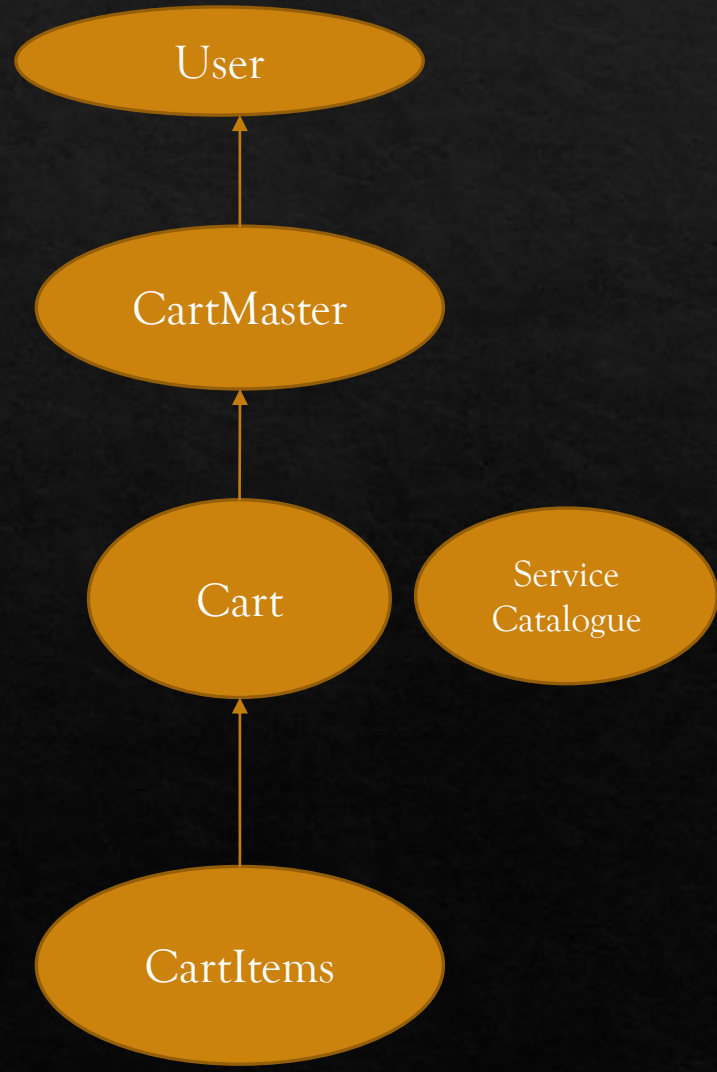
Since Parks are Single Table Operations, and No Other Actions are Required in S1 No Service Definitions are needed. We Added Only API Logging.

S1 Required EP Collection #2

Cart

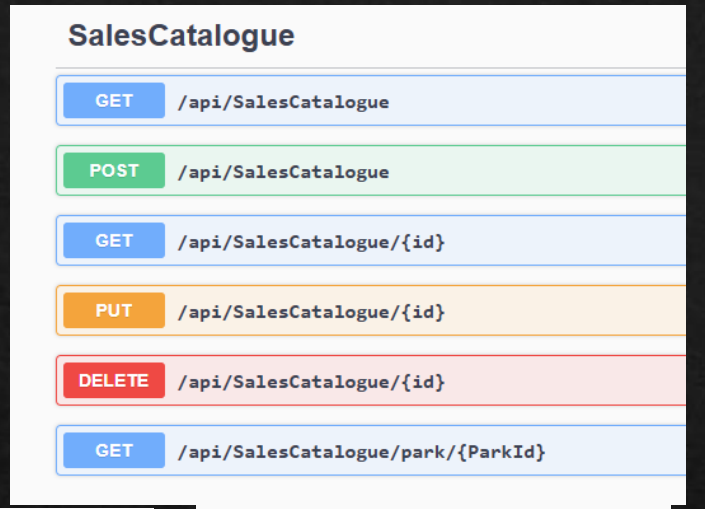
Our Cart Model is Sophisticated… 5 Tables

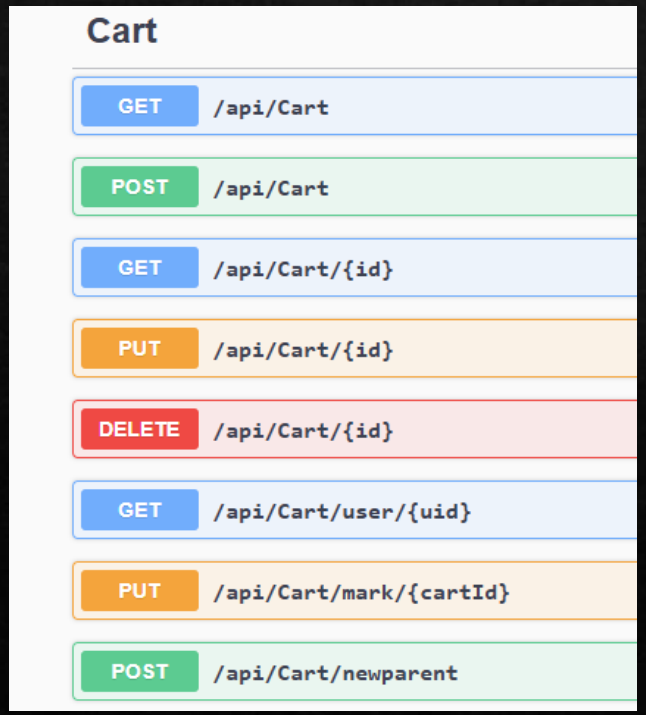
- CartMaster Contains the Entire Sales Record of a User. Every CartID in any state (Booked, Cancelled, RemovedFromCart).
- Cart is the Parent of CartItems, and Contains Summary information about a transaction, the User Making it, the number of adults of children involved, and a date range.
- Cart contains The TransactionID for all payments, and the Booking ID for the Reservation Made.
- Cart Contains the summary tax information total, and by splits local vs state.
- CartItems contains line item detail with User Quantities, Tax Information, etc.
- One User can have many Carts(IE transactions), and many CartItems per Cart.

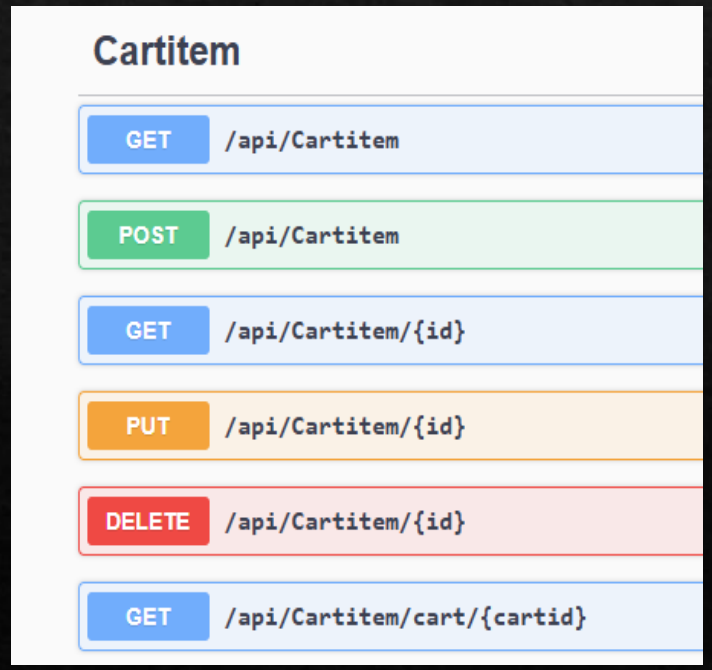


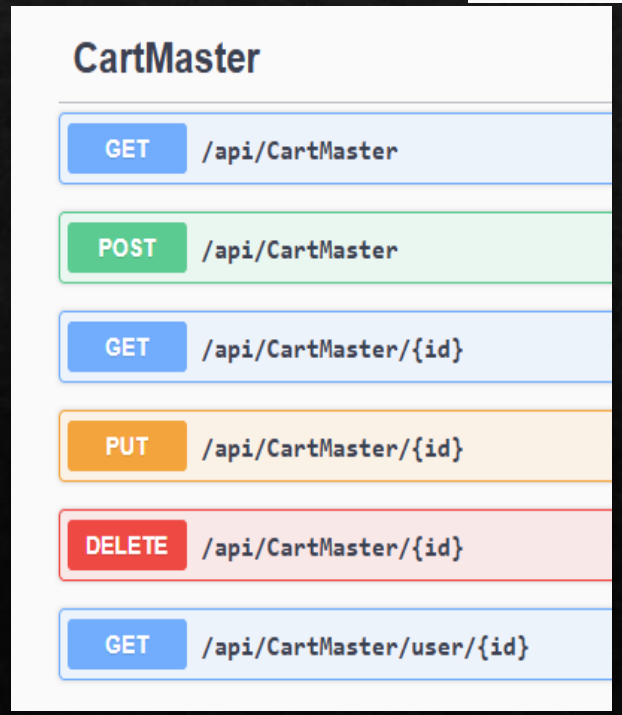
Swagger (34 Endpoints Driving Cart)

- User,Cart,CartItem, CartMaster
- User Updated to have its CartMasterID





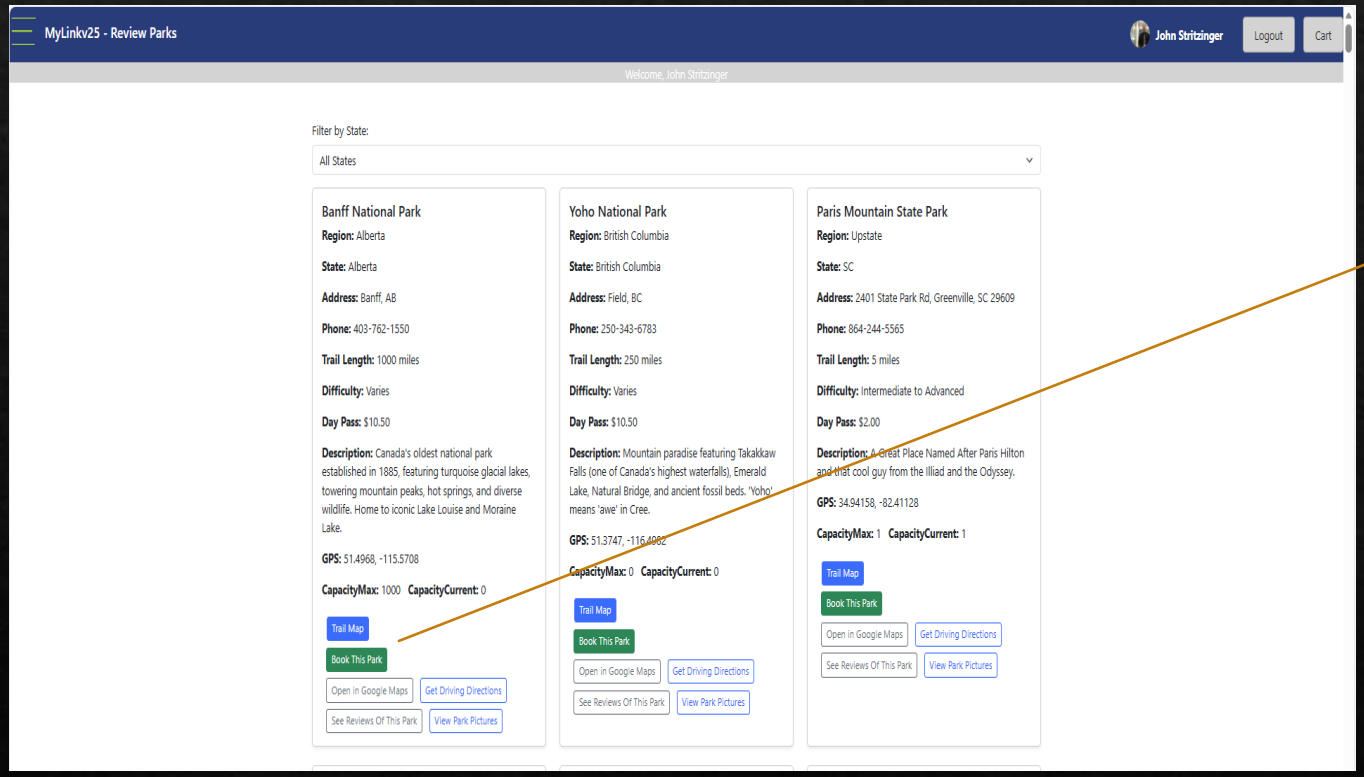


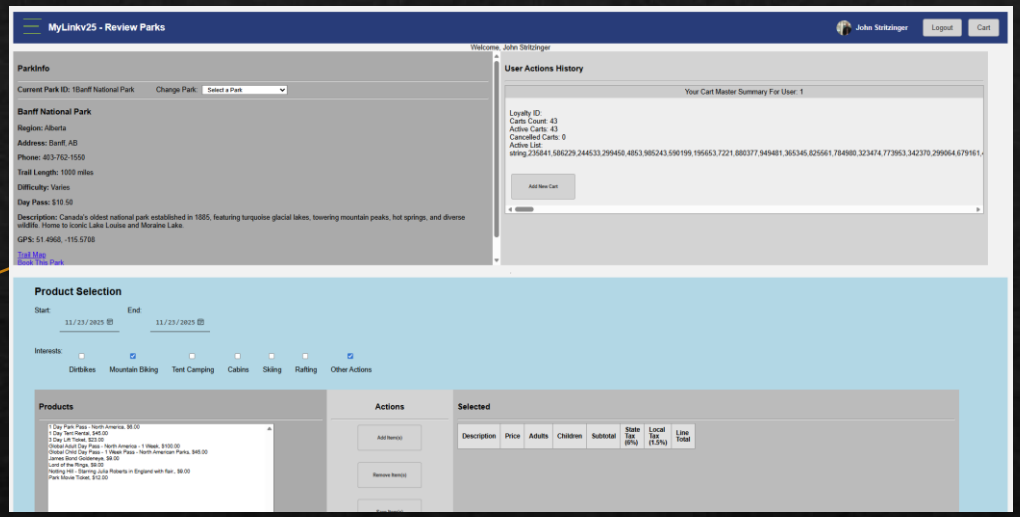


Main Screen for APP.....

- Booking 555.html(After S1.Demo) which has Taxes.
- Booking 55.html delivered day of demo to Production which is working but without taxes and line item based Adult/Child combos.
- Booking5.html was working for S0 demo previously offered and recorded.

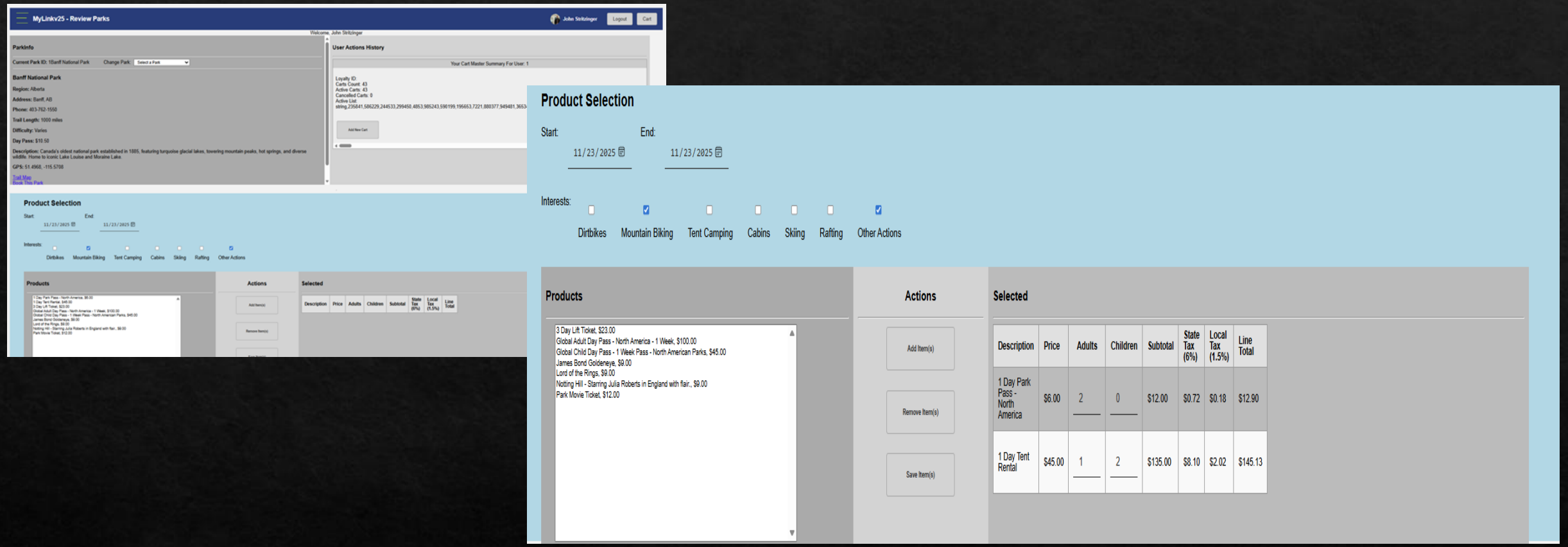
Booking5 and Variants is Very Complicated





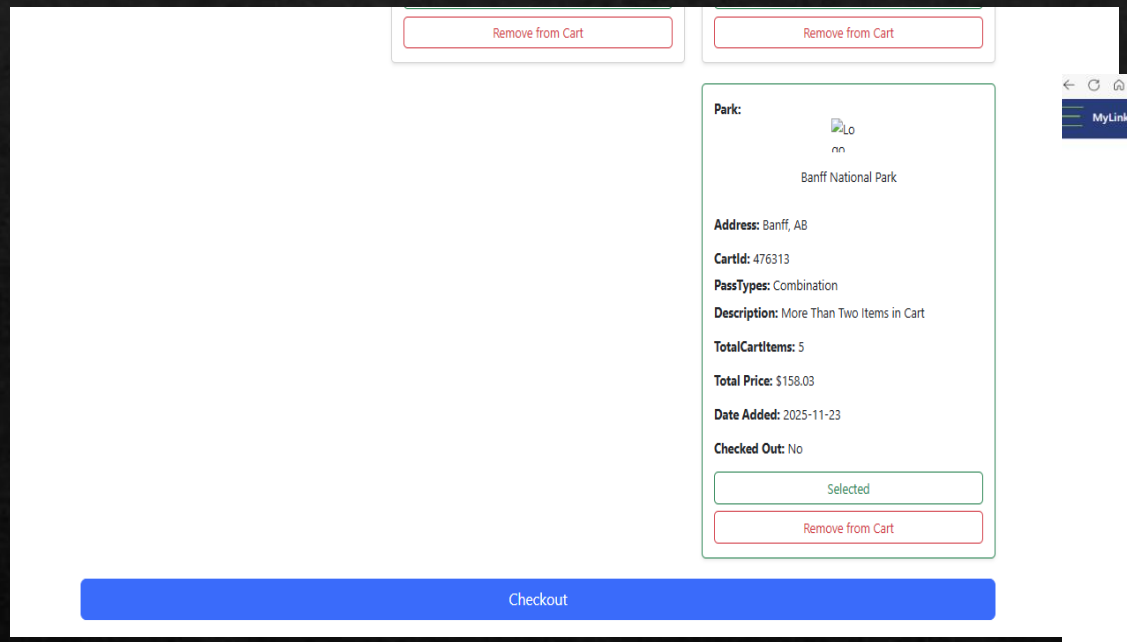
Booking 555 is new since S1, and supports more complicated cartItems but doesn't look as nice right now as original. Please consult pre-recorded video for info as part of submission.

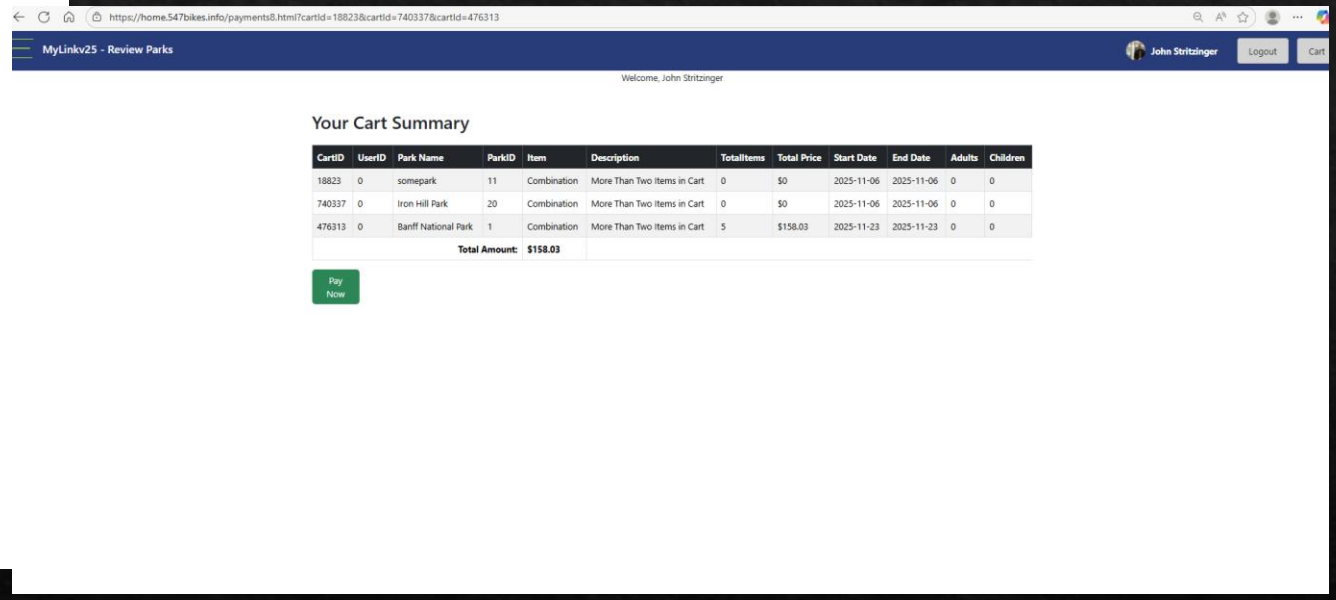
Deep Dive on Main Screen for App Book555.html



On Load -> We Call Parks(Get), CartMasterByUserId(Get), ProductCatalogue(Get), And Cart(Post).
After Products Are Selected, and Values Selected -> the UI does an internal calculation of SubItems Totals.
The Product List is Filtered for the Park Selected by the UI.
After Selection (The UI runs a ForLoop to insert CartSubItems) (AddItemsToCart Function in Javascript).
Creating a Cart Transaction is not a Service. It is a Single Record Post to Cart with the UserID. It requires no C# logic.

From Selected Items to CartReview….





We have the appropriate flows and amounts moving to Cart Summary but the biggest issue is after iterating through the cart subitems, a CartTransaction(Parent) has to be updated… we intend to make this a service soon… but it wasn't required for S1. This will be from UI to Cart EP for just that purpose soon. We are doing this in Javascript during CartTransaction review now.

S1 Required EP Collection #3

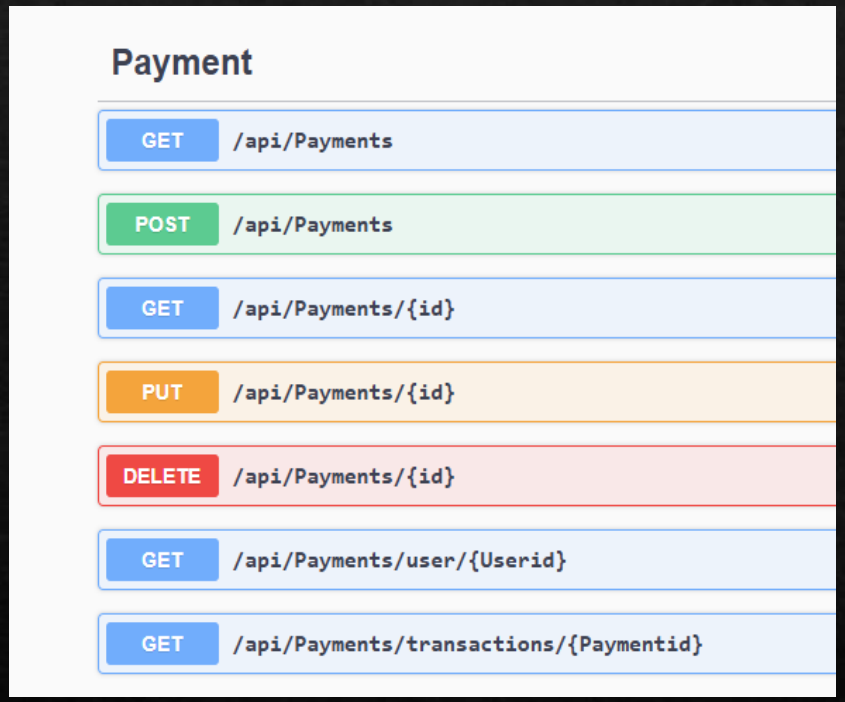
Payment

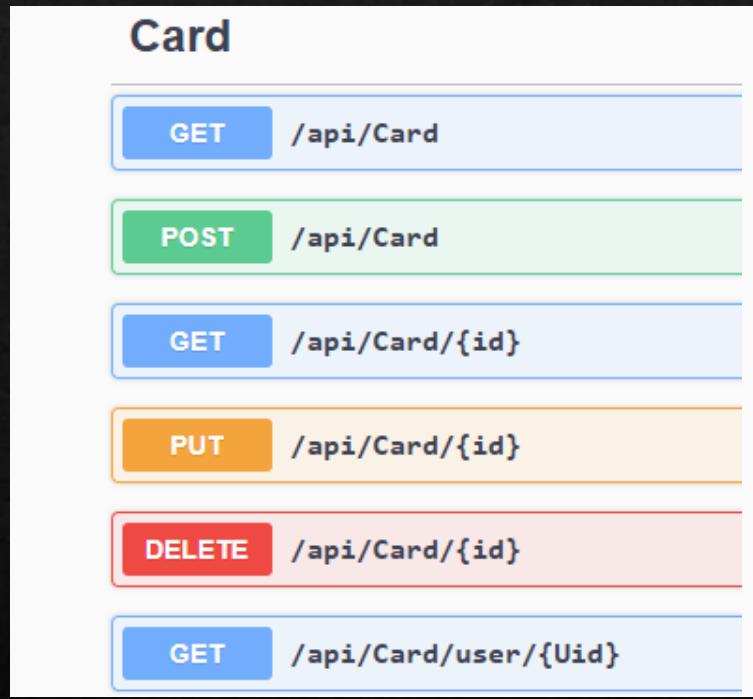
Payments

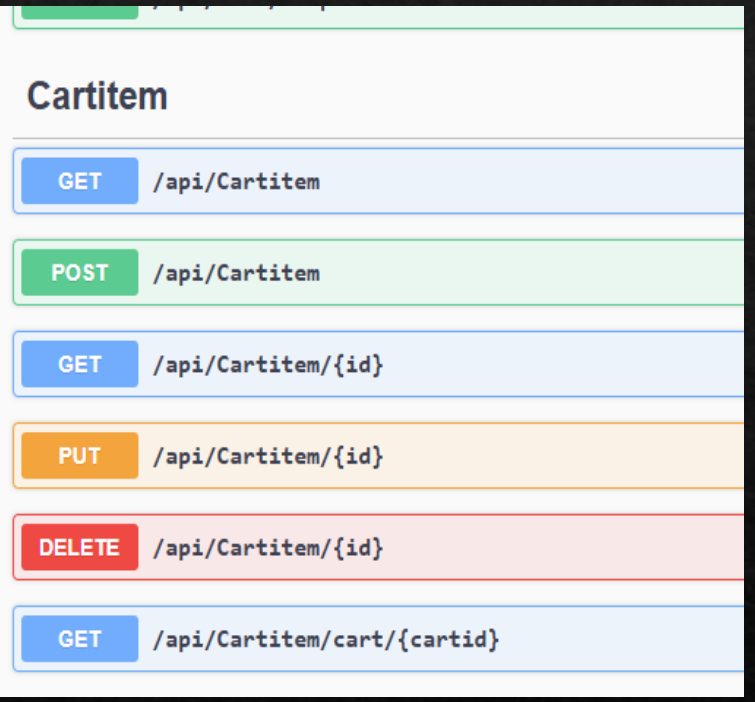
- Select CartID For Payment From Many.
- Pay For A Group of CartIDs by Credit Card
- Add A New Credit Card.
- Create a Transaction
- Create a Booking Following Payment.

- Requires Access to Cards, Payments, Cart, Booking EPS (23+ EPs)

Swagger (Payments, Cards, Cart)







We believe there are 19 EPs needed for Payments.

S1 Required EP Collection #4

Booking

(S1)Booking Process

- We think the design of the Rubric was incorrect as Booking Doesn't occur until after payment which was agreed by the staff.
- Parks->Cart->Booking is the process both in the rubric, in the CG UI provided, and in the UIs built by our team.
- However Booking itself is a single table operation supported by 4 CRUD endpoints.
- No Services were required in S1 for Fulfillment, and therefore the only requirement was 4 CRUD Endpoints (Get, Post, Put, and Delete).
- We have a SQL Lite Table with the Following Fields. Our DB Model is sufficient, it is however difficult to populate correctly.


```
"bookingId": 105,
  "uid": "1",
  "billingTelephoneNumber": "8041111234",
  "creditCardType": "Discover",
  "creditCardLast4": "3321",
  "creditCardExpDate": "04/2026",
  "quantityAdults": 1,
  "quantityChildren": 0,
  "customerBillingName": "John S. Stritzinger III",
  "totalAmount": 32.24,
  "transactionId": "VAD852532684",
  "parkId": 1,
  "parkName": "Banff National Park",
  "cartid": "547981",
  "reservationtype": "Biking",
  "reservationstatus": "Active",
  "reversetransactionid": null,
  "cancellationrefund": null,
  "cartDetailsJson": "[547981]",
  "totalcartitems": 1,
  "reference": null,
  "subReference": null,
  "adults": 1,
  "children": 0,
  "resStart": "2025-11-19T00:00:00",
  "resEnd": "2025-11-19T00:00:00",
  "tentsites": 0
}
```

Booking Architecture

Active Reservations

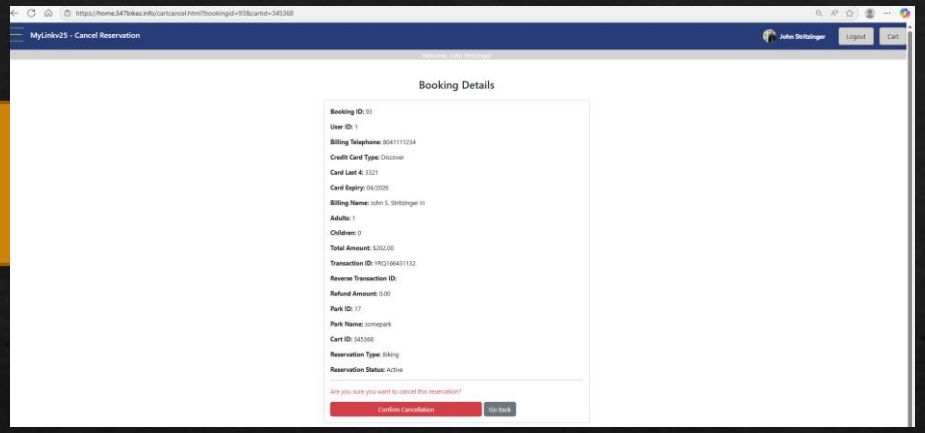
BookingId	UserId	ParkId	CartId	Park Name	Booking Start	Booking End	Adults	Children	Total ($)	Transaction ID(s)	Booking Type	Booking Status	Actions
93	1	17	345368	Hunting Island State Park	2025-11-09T00:00:00	2025-11-09T00:00:00	1	0	202.00	YRQ166431132	Biking	Active	Review Cart Cancel
94	3	159	724279	Skyline MX Park	2025-11-10T00:00:00	2025-11-10T00:00:00	1	0	66.00	SSV604832283	Biking	Active	Review Cart Cancel
100	1	11	967694	Paris Mountain State Park	2025-11-19T00:00:00	2025-11-19T00:00:00	1	0	0.01	VXD621433483	Biking	Active	Review Cart Cancel
96	1	20	503433	Iron Hill Park	2025-11-09T00:00:00	2025-11-09T00:00:00	1	0	280.00	KZI672834021	Biking	Active	Review Cart Cancel
98	1	159	591405	Skyline MX Park	2025-11-10T00:00:00	2025-11-10T00:00:00	1	0	66.00	CXS598875733	Biking	Active	Review Cart Cancel
5	1	999	13	Test Park	N/A	N/A	null	null	18.00	XNO664345003/ ABC456789	Biking	Admin Review	Review Cart Cancel

Grid Architecture Using Bootstrap Layout with a Get Call to /api/Booking

Since Booking Alerts to Users are Extra Credit They were not required in S1.

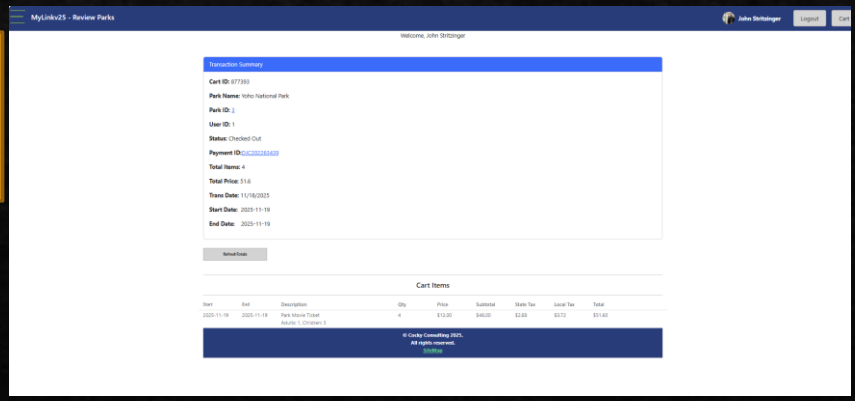
Cancel

Call to cartcancel.html



Cancel Runs a Post Operation to Set a Soft Delete on the Record. It also does a POST to the Transactions Table for the Reverse Amount but that is a S2 REQ.

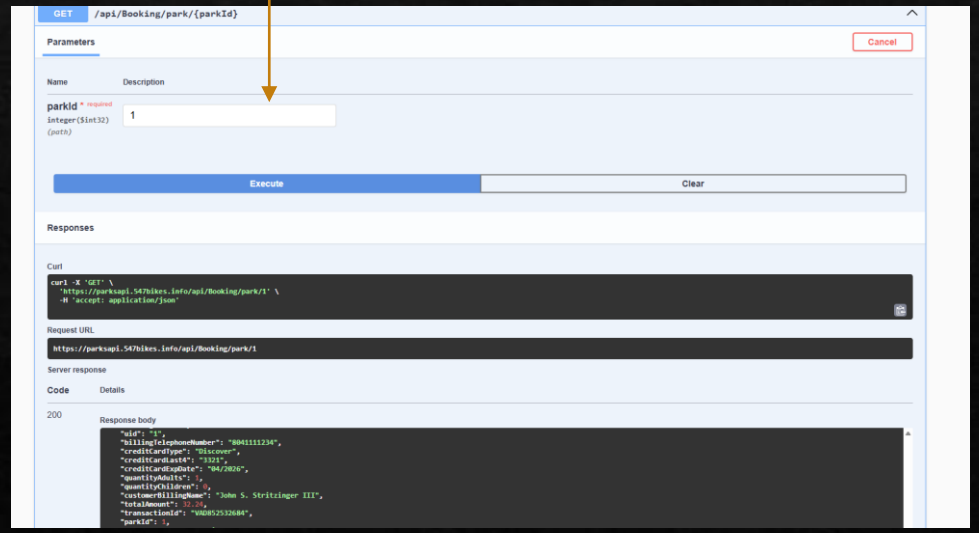
Review Details



Review Details Is a Further Get Operation to Cart/CartItems.

SWAGGER ENDPOINTS BOOKING

- CRUD ON ID 4 EPS
- 2 SPECIALITY ENDPOINTS BY PARKID



SWAGGER ENDPOINTS FOR PAYMENT TRANSACTIONS FROM BOOKING

A CANCELLATION HAS TO UPDATE PAYMENTS WITH THE REVERSE AMOUNT. THIS WAS A S2 REQUIREMENT.

Solution Review

01

OPTIONS FOR SERVICE DELIVERY

02

SERVICE ARCHITECTURE DESIGN CHOICES

03

SERVICE ARCHITECTURE VS RUBRIC

RubricS1 vs Actual

- S1 Rubric
  - SQL Tables Required-> 0
  - EPS Demonstrated ->5
  - EPS In Rubric for S1 ->12
  - Services Required->Zero Only EPs.
  - Repositories Required -> 0
  - DTOs Required ->0
  - Unit Tests Required-> 12.
  - UI Required-> No
- Actual
  - SQL Tables > 30 SQLITE TABLES Modeled
  - Demonstrated > 150 Eps
  - S1 Required -> 70 (8Parks+14Pay+34Cart+14Book)
  - Services Provided 2->(Auth, APILog) with many framed
  - Repositories Provided -> 0
  - DTOS Provided ->1 (User)
  - Unit Tests: 108+ End To End Transactions with Simulated Payments.
  - UI Provided (2) -> App(IonicReact), And Full JS with Auth from GhostPortal.

C# Demonstration

- All the EPS in the System > 150 using Entity Framework Core with SQL.
- Standard IF Blocks for Puts support partial Updates of a Record.
- APILogs demonstrate capability to Build Server Side Services from Back to Front.
- We showed our capability to use Azure Fundamentals by Building a DevOps Dashboard.
- We Showed our capability to use Azure Fundamentals to build a static webapp.
- We showed our capability to use Azure Fundamentals to build an Azure API running SQLITE.
- We mocked up a C# installer for basic parks which we did not use extensively but was in the project.
- We built triggers and fixed issues with User/UserProfile which was an action from Spring 21.
- We Used Reverse Engineering to Build Models in the Model Directory.
- Reversed Engineered Tables create a Context which builds a C# entity from each endpoint.
- Models show the basics of nullables, and transparency.
- Provided a C# Installer which showed capability of Building Menus on the CLI run EF SQL Scripts, Loops and Lists which was built in 547 in Fall 2024.
- We intend to build many services for fulfillment however most things can be done by the controller.
- We have not done an assessment on the CG App requirements but it should be nearly identical to our own logic which we have laid out.

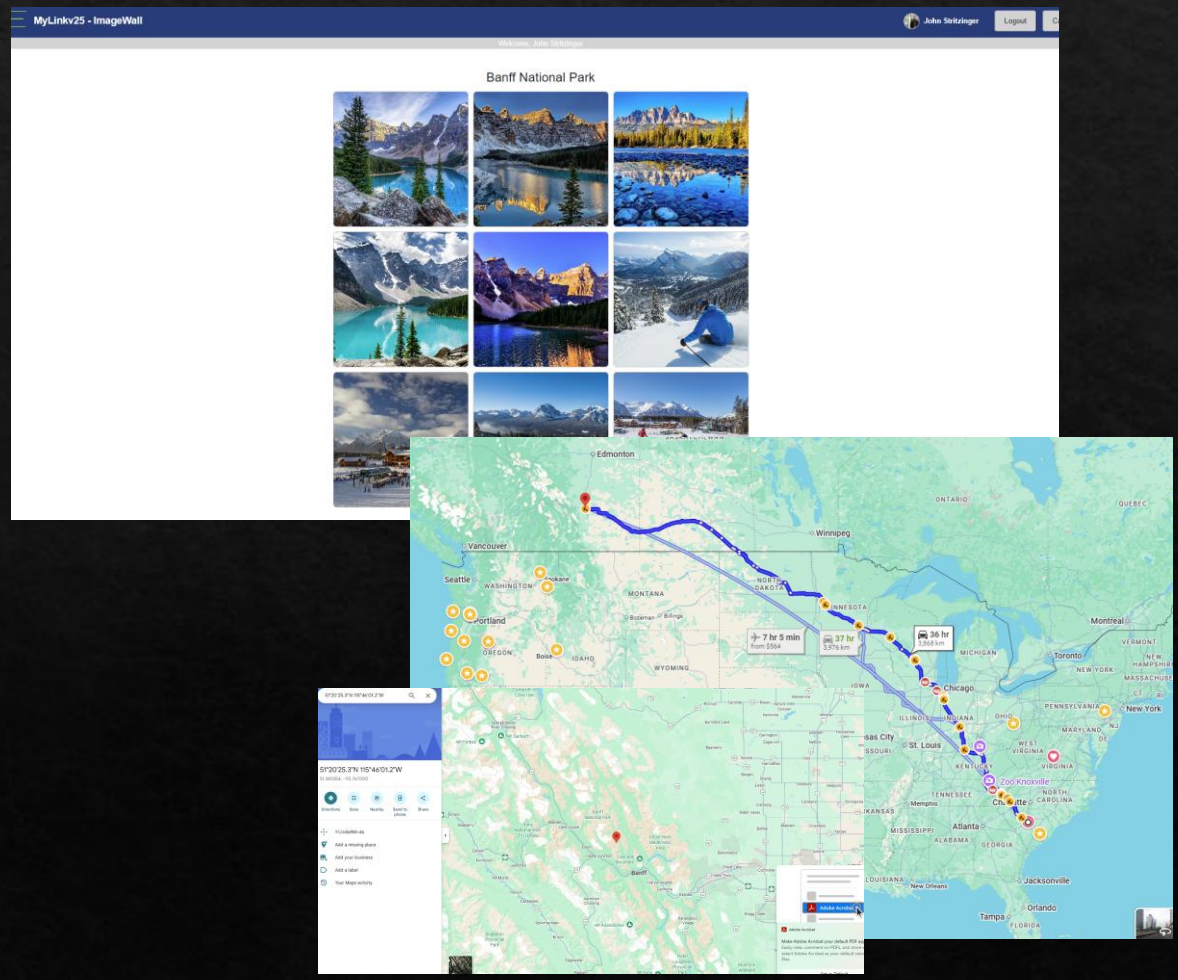
Teamwork in this project

- Capgemeni has provide a UI by L. Elwood which needs to be integrated for Final.
- Capgemeni has provided a development and testing methodology which is very powerful.
- G/A has contributed specific Models for Managing PLL(Page Level Locking) and UserState functions.
- CTS has provided a hosting model which was developed on Google Cloud since 2018 with production hosting (DEVSERVER).
- Team2 for this fall semester was involved in the design of the database architecture and DEVOPS model we used to date. (SQLite flat file model).
- Team1590 has provided all the User state information, tables, and Templates for Rapid Endpoint Development or Participating bymyself would not be possible.
- Team1590 spent significant time working on the Ghost Portal Navigation and Screens in support.

- This project team is a MATRIXED TEAM using other Student Resources owned by other various business units.

Major Upgrades From Rubric

- Better And Full Featured Park for any activity.
- Full WebUI, and AppUI React.
- Security
- Product Sales Catalogue is as significant as 590Team Project.
- Parks Picture Wall being driven from Database.
- Driving Directions
- Map Links
- Working UI/Auth UserProfile from 590.
- Granular Sales Options by Park, Global or State
- SecurityCenter.html
- Universal Cart.
- CartMaster UserProfile Link.



WrapUp And Open Issues

Team on Track to S2/Final But User Demo Already Complete.

Final Presentation

- We effectively gave our final presentation, with many backoffice functions needing to be cleaned up before final code submission.

Gaps to Final Presentation.

- Our UI, and App are finished to our backoffice. (UI Done).
- Our Backoffice services and notification are not yet complete. This is a S2 requirement.
- We have an issue with ZeroCarts, which need to be removed before moving from Book555 to cartreview.html. This was not a S1 requirement.
- Cartreview.html, and payments need updated grid fields showing the right dates, and users which are in fact properly stored in the database. This was not a S1 Requirement.
- CG integration is unknown at this point in time and was not required for S1.