DEVOPS MODEL

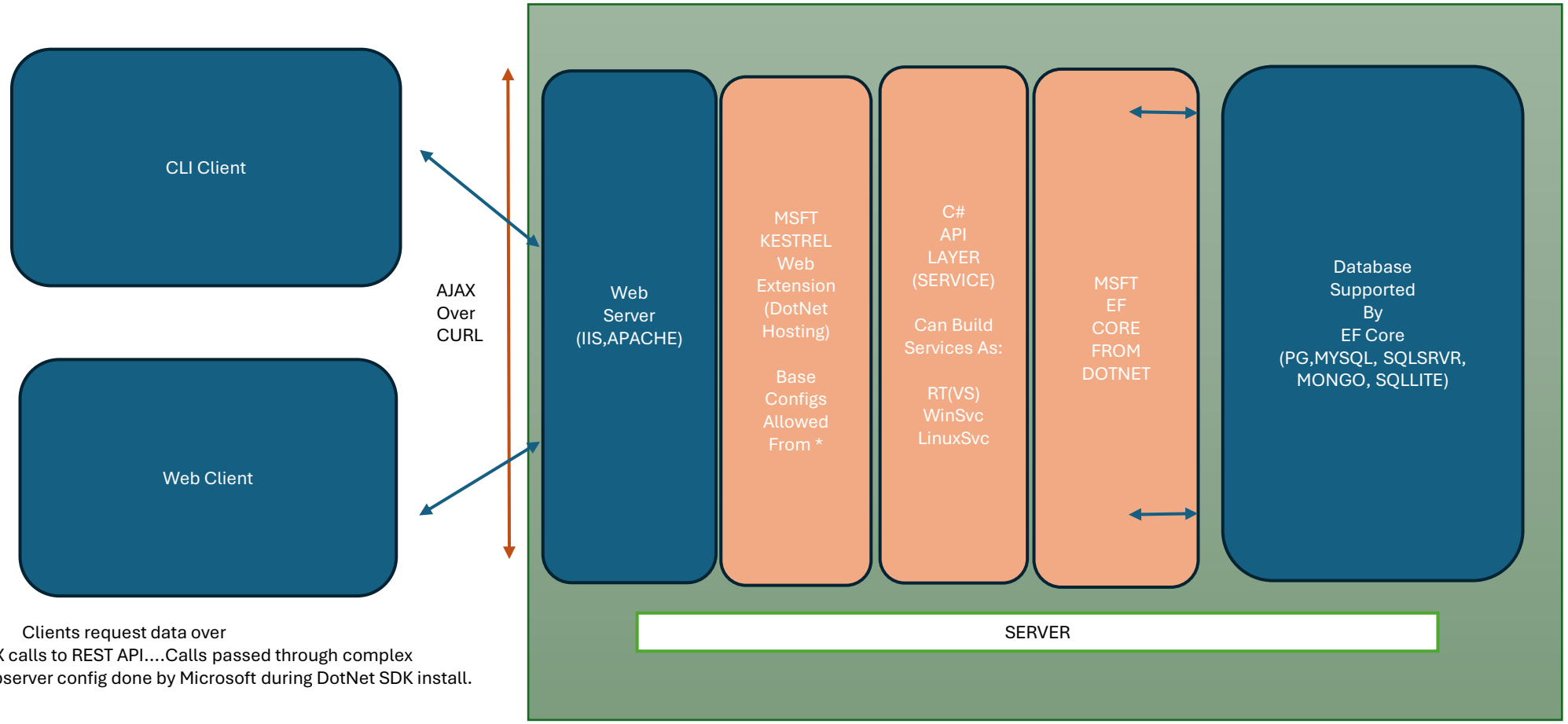
Proposed

V8.0

This class is designed….

- To test command line capabilities in C#. (Phase I).
- To test web capabilities in C# (Phase II).
- Last Year:
  - Teams last fall spent most of their Phase I demo time on simulating credit card transactions to their carts as this is very easy to use all of the C# rubric items.
  - Seeding data is involved in insert data/tuples into your database so that it populates initial data. This can also be done via C# and uses basic loop structures, lists, and C# datastructures which is most of the Phase I requirements.
  - However last year our team decided to demonstrate all the C# rubric requirements in the installer, and focus on our efforts on the web requirements so you don’t waste development time on a useless CLI. We didn’t build a CLI at all initially(I did for final only). We demonstrated our web tool in Phase I.

High Level of the Process.... Hosted



1) Clients request data over AJAX calls to REST API....Calls passed through complex Webserver config done by Microsoft during DotNet SDK install.

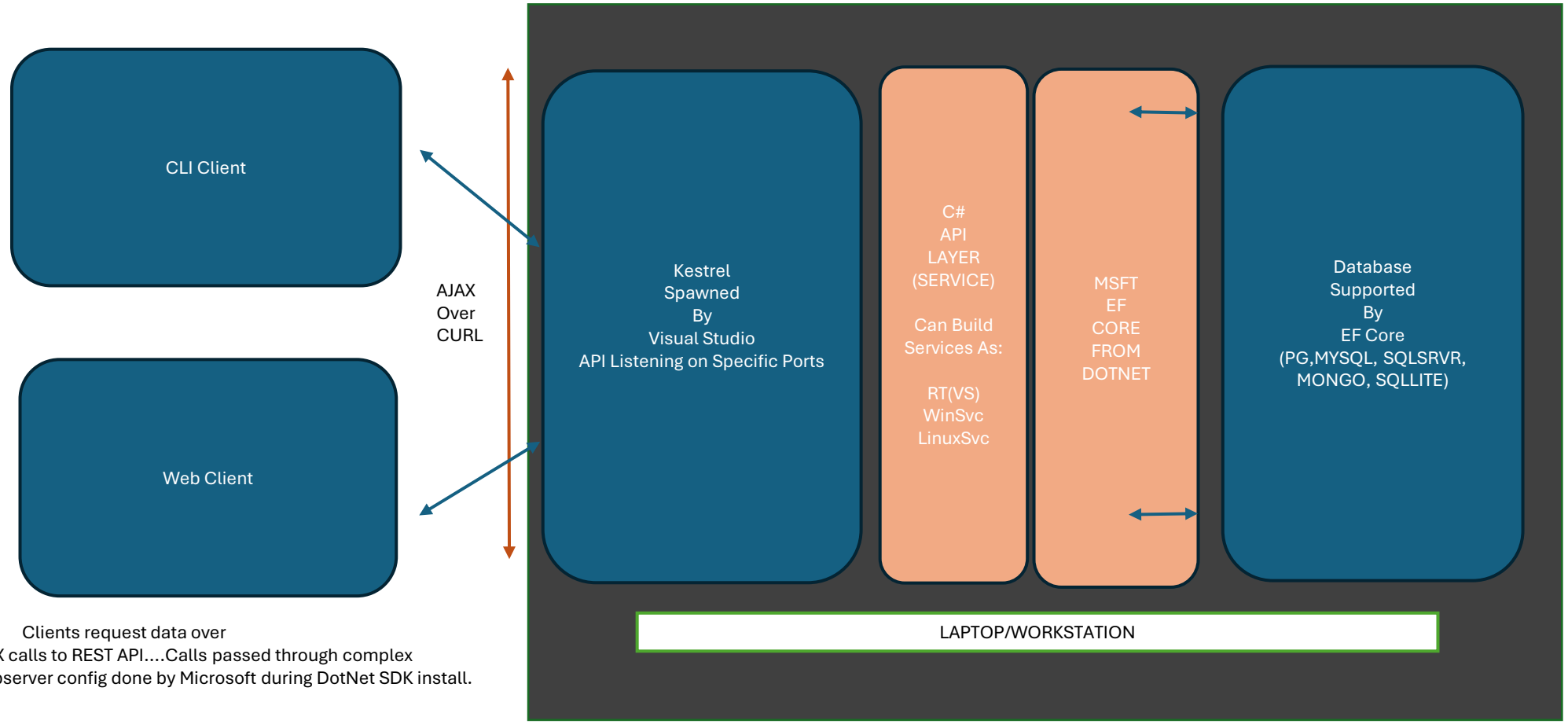
2) The Webserver gets the request, and proxies requests to the API layer on any port required. 80/443-> API PORT. CURL Terminates on C# Service.

3) The API communicates over EF to the Database.

APIs for this project are simple. They don't require joins.... because they are user driven. So you essentially have one API EP for every table. Its effectively prebuilt SQL commands.

The cart service is a bit more complicated as it requires temporary space in some form.

High Level of the Process.... Local



1) Clients request data over AJAX calls to REST API....Calls passed through complex Webserver config done by Microsoft during DotNet SDK install.

2) The Webserver gets the request, and proxies requests to the API layer on any port required. 80/443-> API PORT. CURL Terminates on C# Service.

3) The API communicates over EF to the Database.

APIs for this project are simple. They don't require joins.... because they are user driven. So you essentially have one API EP for every table. Its effectively prebuilt SQL commands.

The cart service is a bit more complicated as it requires temporary space in some form.

Decisions to Make

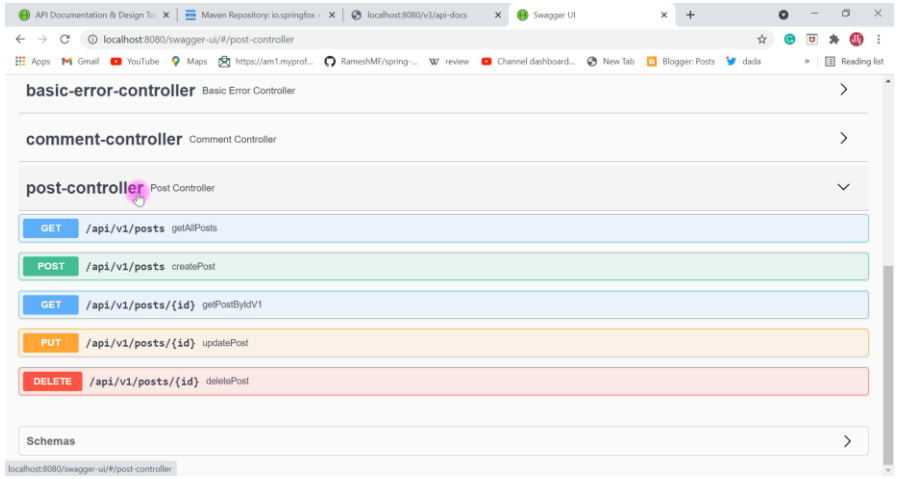
- Code First Vs Database First
- Database Type
- Development Model
- Hosting Plan
- API Management
- Team Ownership Jobs, etc

Code First vs Database First

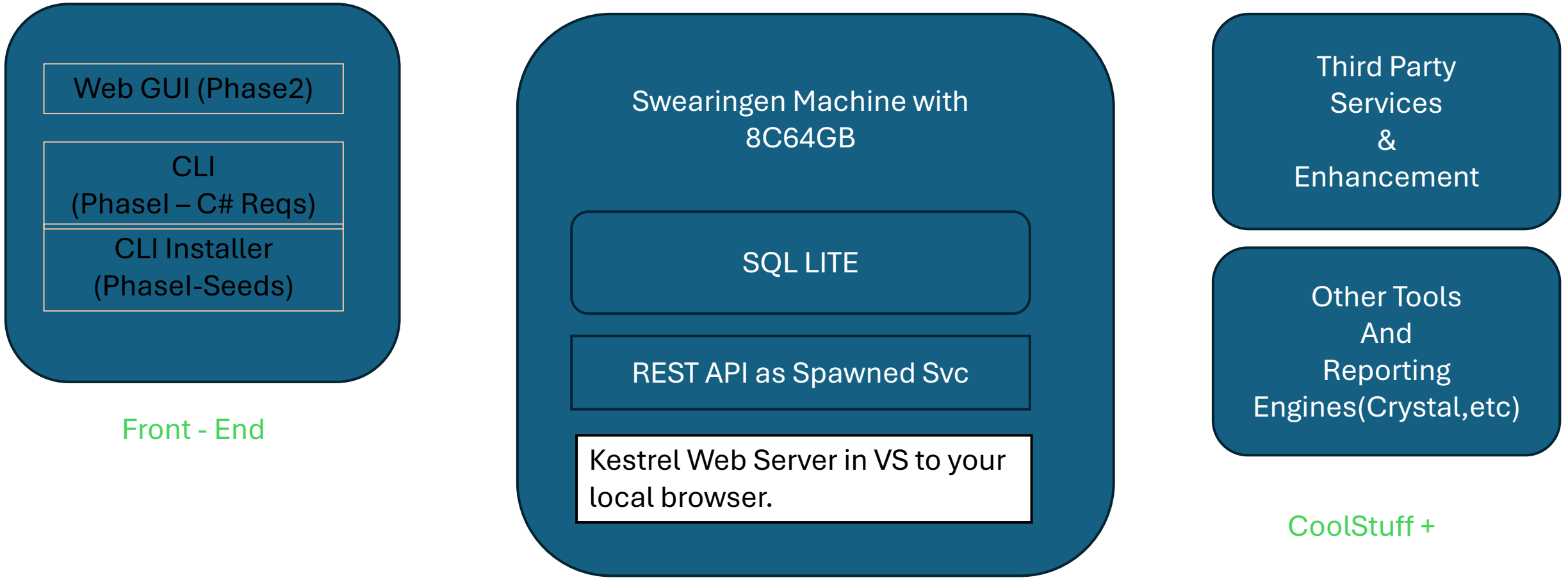
- In Code First… you write C# code, and it creates tables using your connection string…. assuming you give it an Admin Login….
- In Database First… you build your database architecture first, and then have Visual Studio login into it and generate code for the table models (your gets and sets).
- https://builtin.com/articles/code-first-vs-database-first-approach

When the APIs are built....and running

- The Webserver API outputs a webpage with each endpoint called Swagger.
- SWAGGER allows you to check each endpoint separately post and retrieve data... if swagger works your endpoint works. Swagger does effectively run SQL commands against a dataset, list, or some other MS supported data structure.
- In class they will also demonstrate Postman which is fancier but runs as a thick client on your laptop.
- Last I year I found a VSCODE package called "REST API" endpoints which allows you to send gets, and puts to the API with easy to understand output. Search for this as a plugin.



Option 1a – SQL Less - ProjectDesign

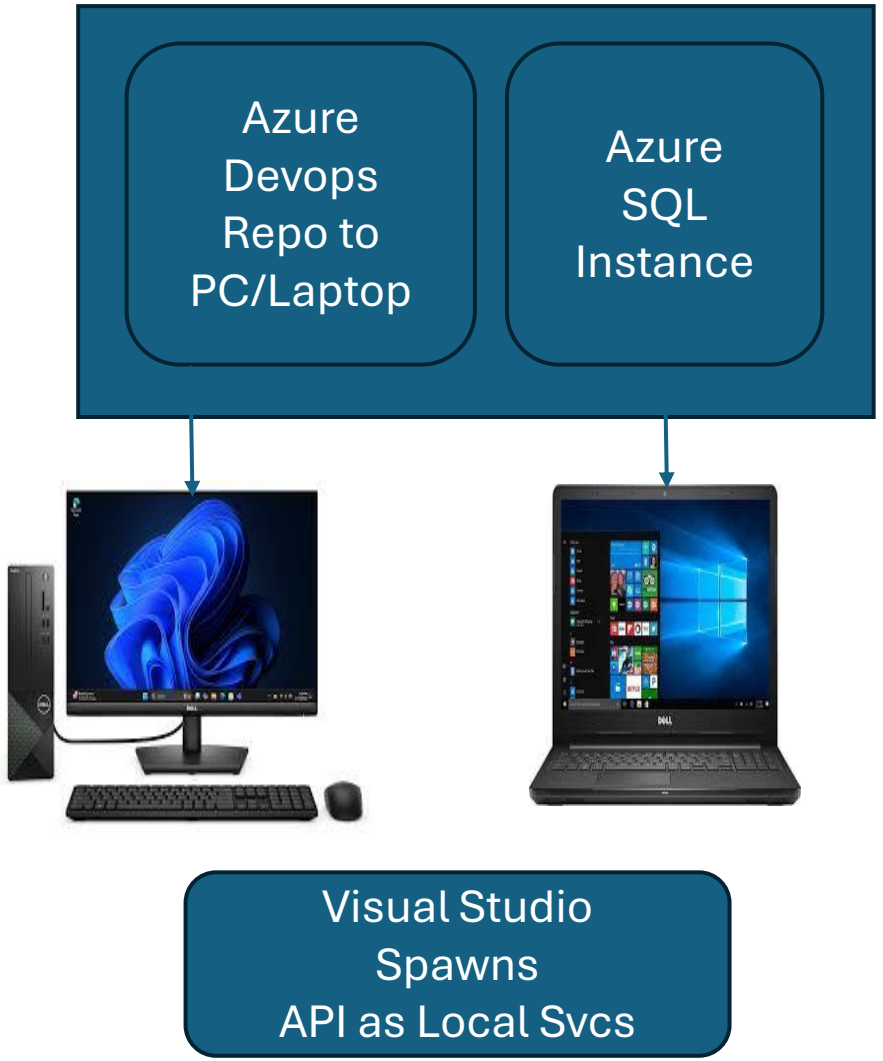


Option1a – Local Devops with Flat File

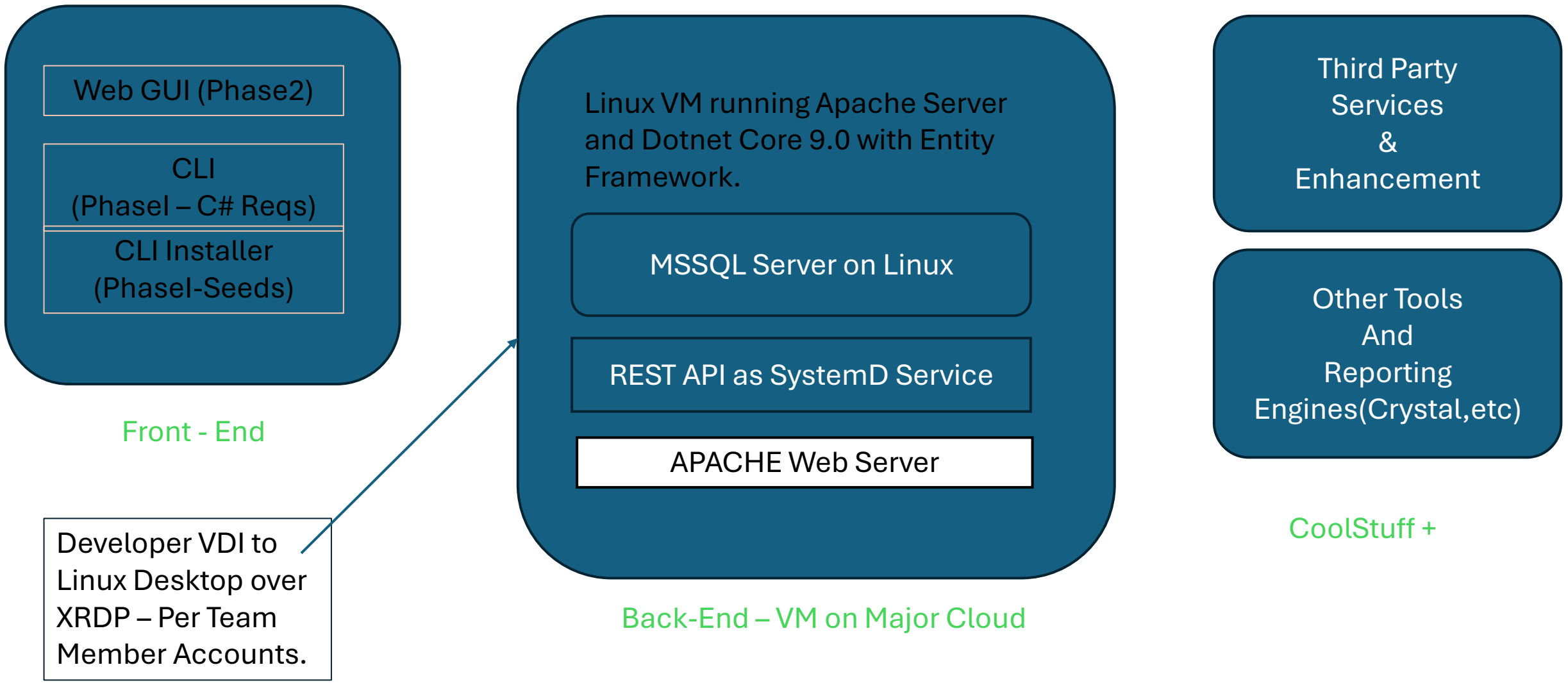
- Project is created in Azure DevOps via one of team members Student Accounts. Permissions granted to team to update.
  - Code is stored in Azure DevOps or Github.
    - (Last Year CG didn't care)
- Developers synch project using Visual Studio Git extensions to local machine/repo as a "checkout process".
- Updates run locally, and posted as changes to the Master Repo and Repo owner approves them for posting.

Option 1b – Just Host a Cloud Database

- Its possible as Tayyaba mentioned yesterday to just host a cloud database instance, and run API's locally during demonstrations.
- Azure SQL is Free
- Azure Cosmos DB is Free

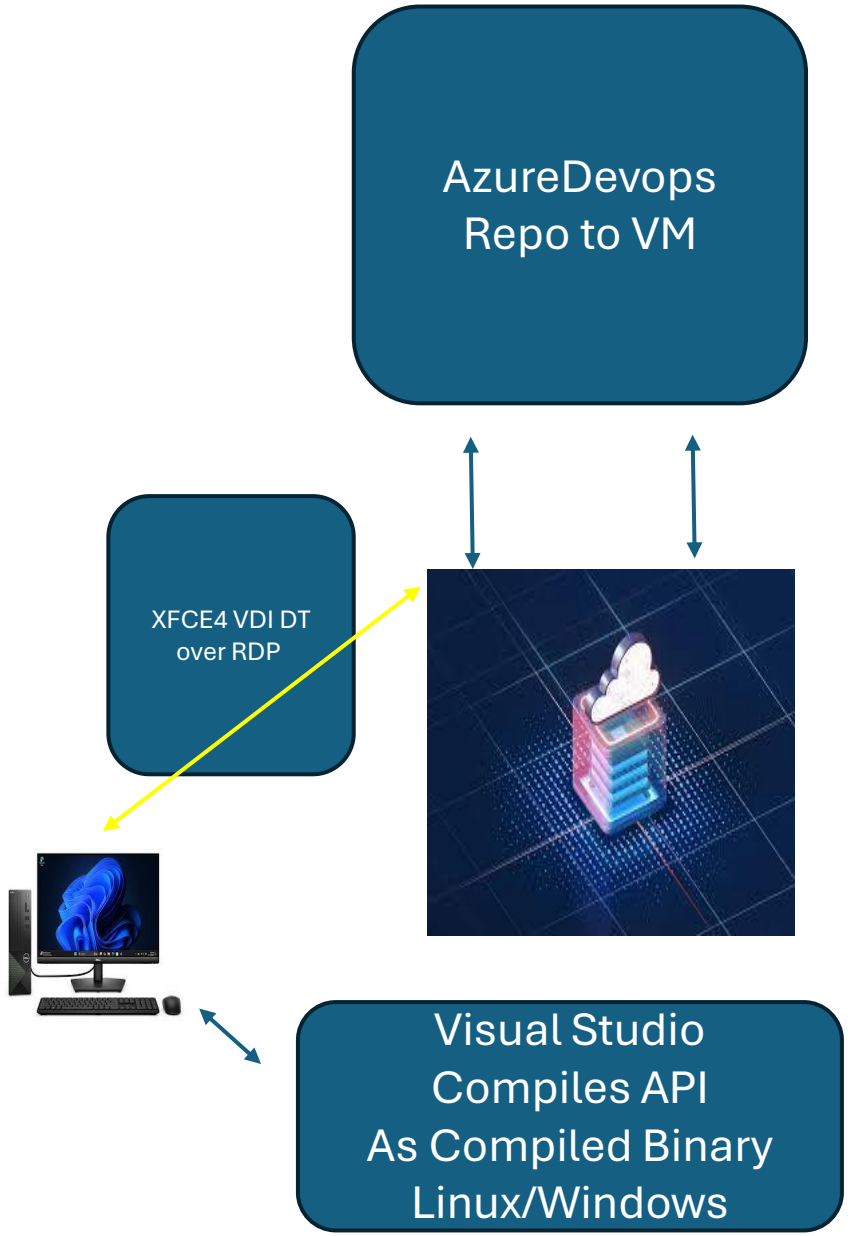


Option 2 - Hosted Design....Recommended...



Option2 - DevOps

- Project is created in Azure DevOps via one of team members Student Accounts. Permissions granted to team to update.
  - Code is stored in Azure DevOps or Github.
    - (Last Year CG didn't care)
- Developers synch project using Visual Studio Git extensions to cloud machine/repo as a "checkout process".
- Updates run locally, and posted as changes to the Master Repo and Repo owner approves them for posting.

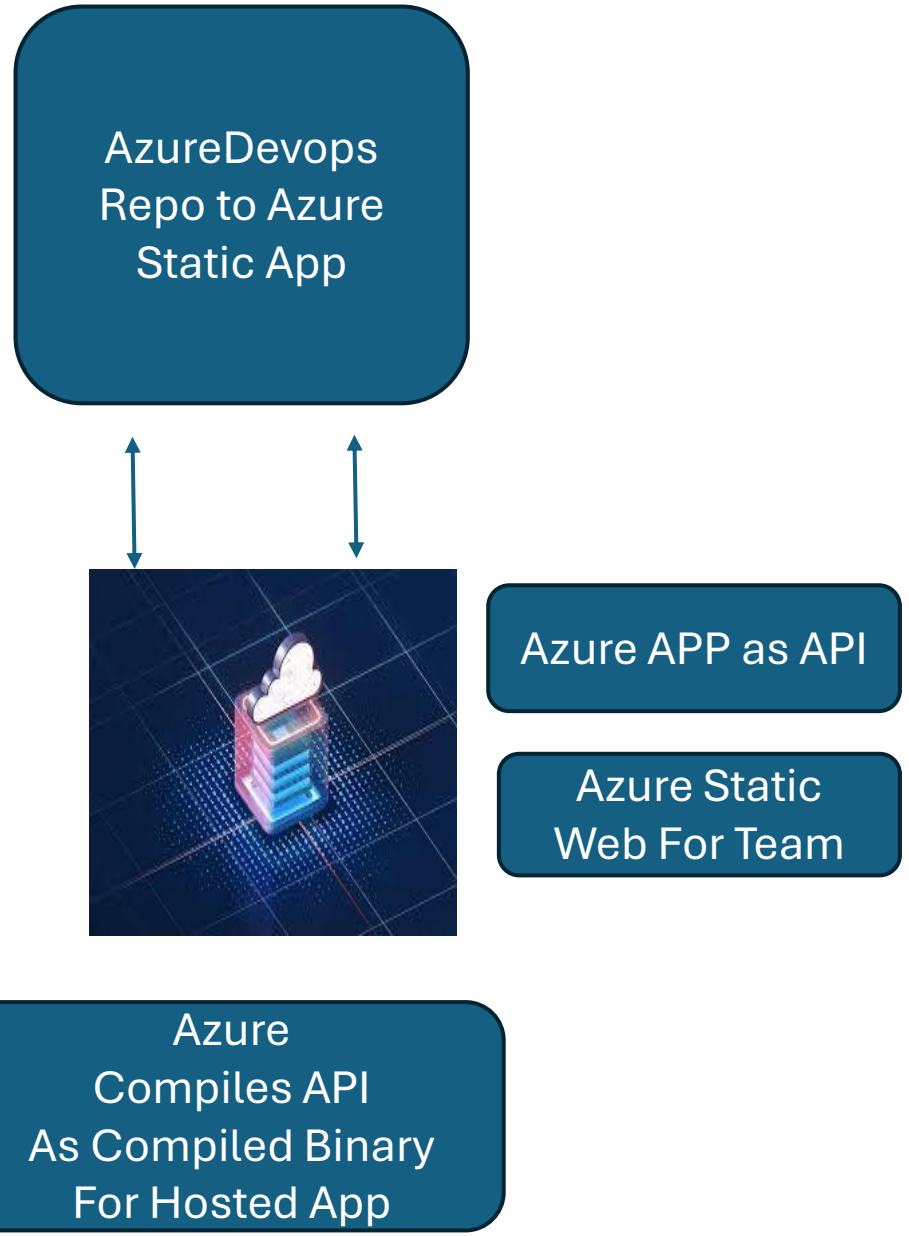


Option3.....Use Azure at 100% for 547

- Host API on Azure like in 590....and use the 590 architecture for this class too with a different project.
- Host Static Web site on Azure like in 590 for the team...and since each student has one....everyone has their own sandbox against a common API layer.....
- Use Azure SQL for the Database.
- Slight Difference in Devops....

Option3 - Devops

- Project is created in Azure DevOps via one of team members Student Accounts. Permissions granted to team to update.
  - Code is stored in Azure DevOps or Github.
    - (Last Year CG didn't care)
- Developers synch project to static apps on Azure from Azure Devops….
- Updates run locally, and posted as changes to the Master Repo and Repo owner approves them for posting.



Option 4 – Campus VMs

- A Campus VM can be approved with Department Head Authorization.
- This could result in a Windows or Linux VM.
- Free to Students, but requires an ask to Ryan Austin.

Option 5 – Docker Desktop

- There are various ways to host docker images on the Internet.
- You can buy a VM with Docker support.
- You can pay Docker for the service.

* This isn’t supported by the University, but Capgemeni approved last year.

Hosting Costs vs Student Accounts

- Option 6 (Free) - Azure SQL Student Database
- Option 5 (Free on Google) – Docker – Requires CapGemeni Special Permission & a VM
- Option 4 (Free) - Campus VM – Requires Special Permission from Dr. Valefar
- Option 3 (Free)
  - Static Apps on Student Accounts – Azure – Free
  - Static APIs on Student Accounts – Free for non vanity URLS.
- Option 2 (Billing Vs Student Base Funds)
  - Google $300 Base Credit For Students
    - Google VM for 2 Mos ~ 50 for the team 2GB RAM APACHE, 50GB.
    - Additional Funds can Be User for Cloud AI Services, GIS Maps, etc.
    - Net Free
  - AWS VM for 2 MOS ~ 150 for the team 8GB RAM IIS, 100GB.
    - Windows VMs but would cost money.
  - MSFT Student Accounts are $100.00 Base Credit
    - AZ VM for 2 MOS ~ 150 – 50.00 Azure credit or 100.00
    - Microsoft Credits are not enough to support AI, or VM's but Static Apps are Free.
- Option 1b
  - Swearingen PC with Hosted Azure SQL Database - FREE
- Option 1a
  - Swearingen PC with SQL Lite - FREE

Major Jobs in this project

- Project Analyst… Keeping the Team Moving towards Goals.
- Webmaster… Responsible for UI for Phase II
- CLImaster…. Responsible for CLI for Phase I and Seeds/Install.
- API-Repo Owner … Responsible for the Code Releases for the API.
  - This is usually the database owner as well.
  - This is usually the data architect as well.
- Portal SSO
  - Getting Logins working, and User information stored in LocalStorage.
- Cart
  - Cart Logic and the Web UI for it are the hardest part of the project.
- Payments
  - Simulating the Processing of a Payment requires up to 4 DB tables.
  - Requires Hashes in Phase II.

Ways to split tasks

- Front to Back (UI Developers and BackOffice Developers).
- By Area of the Code.
- In Small Teams the Project Manager cant not write code like in some team structures.*

- CG/Venkata requires everyone to write code to get a grade.

Considerations

- SQL Driven Notifications
- Long Term Use of Code By Our Team