DRAFT

eRevStr

Ultimate enterprise string reversal system

Document Version: 0.3
Date: 27/07/2020
Author: Nithin Mohan

Agenda

- Introduction
- Requirements
  - High Level Requirements
  - Data & Security
  - Non-functional Requirements
- Proposed Solution
  - User Flow
  - UI Concepts
  - Architecture
  - Risks & Assumptions
  - Future Scope

Introduction

GOAL

- **Who:** User / Employee
- **Where:** Medium to large Organizations or Business Units or Company
- **What:** Building a customizable multi –tenant string reversal platform.
- **When**: During work to reverse a string
- **How:** Subscription based Software as a Service, accessible through a portal

High Level Requirements

- Building a customizable multi –tenant string reversal system.
- Basic Features:
  - Page for Users to enter the input and submit.
  - System should accept the input → validate → reverse → store → return.
  - Basic portal would be required to be provided as part of the platform.
  - Admin portal for onboarding and managing tenants.
  - Registration page for New Customers.
  - User should only be able to view history of previous inputs and results.
- Custom Frontend Support:
  - System should support using custom frontends for tailor made customer/tenant requirements.
  - System should expose necessary APIs to support integration from other systems.
- Each tenant should be able to customize the branding of basic portal.
- Should be able to use vanity urls for accessing tenant specific instance.

Data & Security requirements are outlined in the next slide

Data & Security

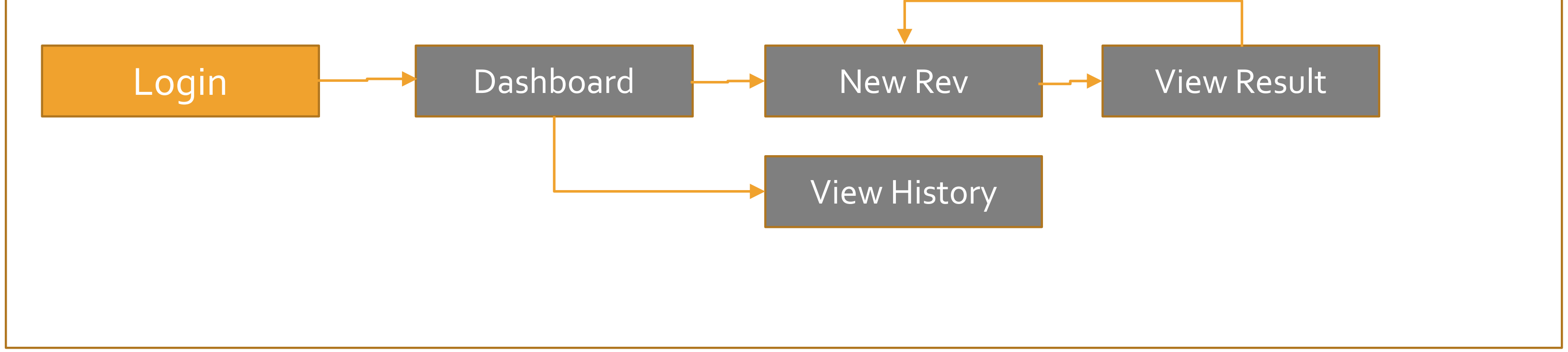
- Data:
  - Configuration Data of each tenant should be stored in isolation. (either as separate tables or database depending on the customer requirements) .
  - Users within Other tenants should not be able to view the records of users within other tenants.
- Security:
  - System should authenticate the user automatically provided user is accessing the system within organizations network infrastructure. (SSO)
  - System should support Integrations with Microsoft AD, Siteminder SSO or Ping Federate.
  - Custom Passwords should be encrypted using Customer Provided Key or system managed key.
  - Following Roles are required:
    - Super Admin
    - Tenant Admin
    - User

Non-functional Requirements

- The system should be scalable to very high numbers of concurrent of users.
- System should be highly available.
- Globally available. Multi region.
- Exceptions should be gracely handled.

User Flow

End User Flow



Login

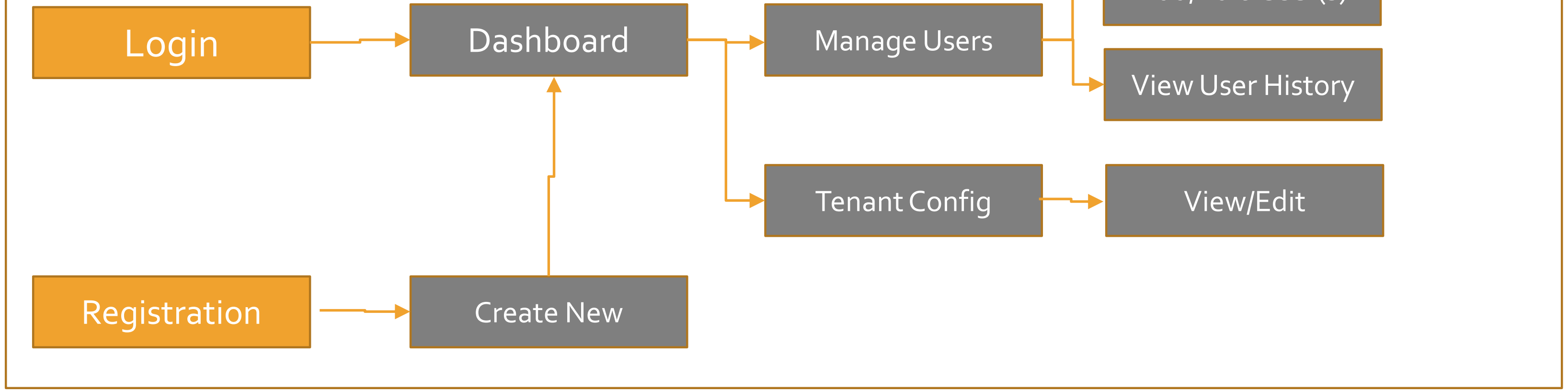
Dashboard

New Rev

View Result

View History

Tenant Admin Flow



Login

Dashboard

Manage Users

Add/Edit User(s)

View User History

Tenant Config

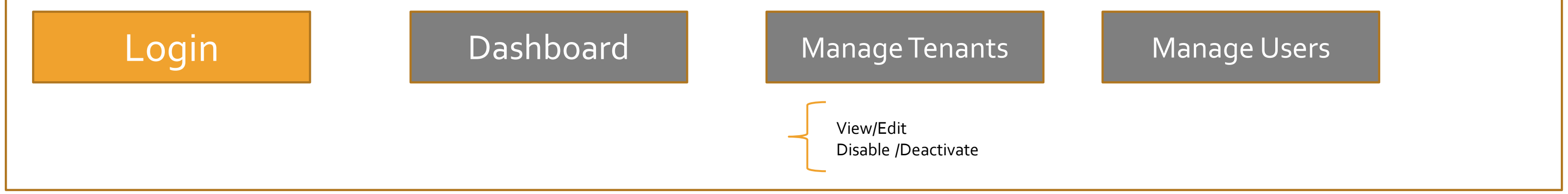
View/Edit

Registration

Create New

Super Admin Flow

All other operations of Tenant Admin can be performed by super admin



Login

Dashboard

Manage Tenants

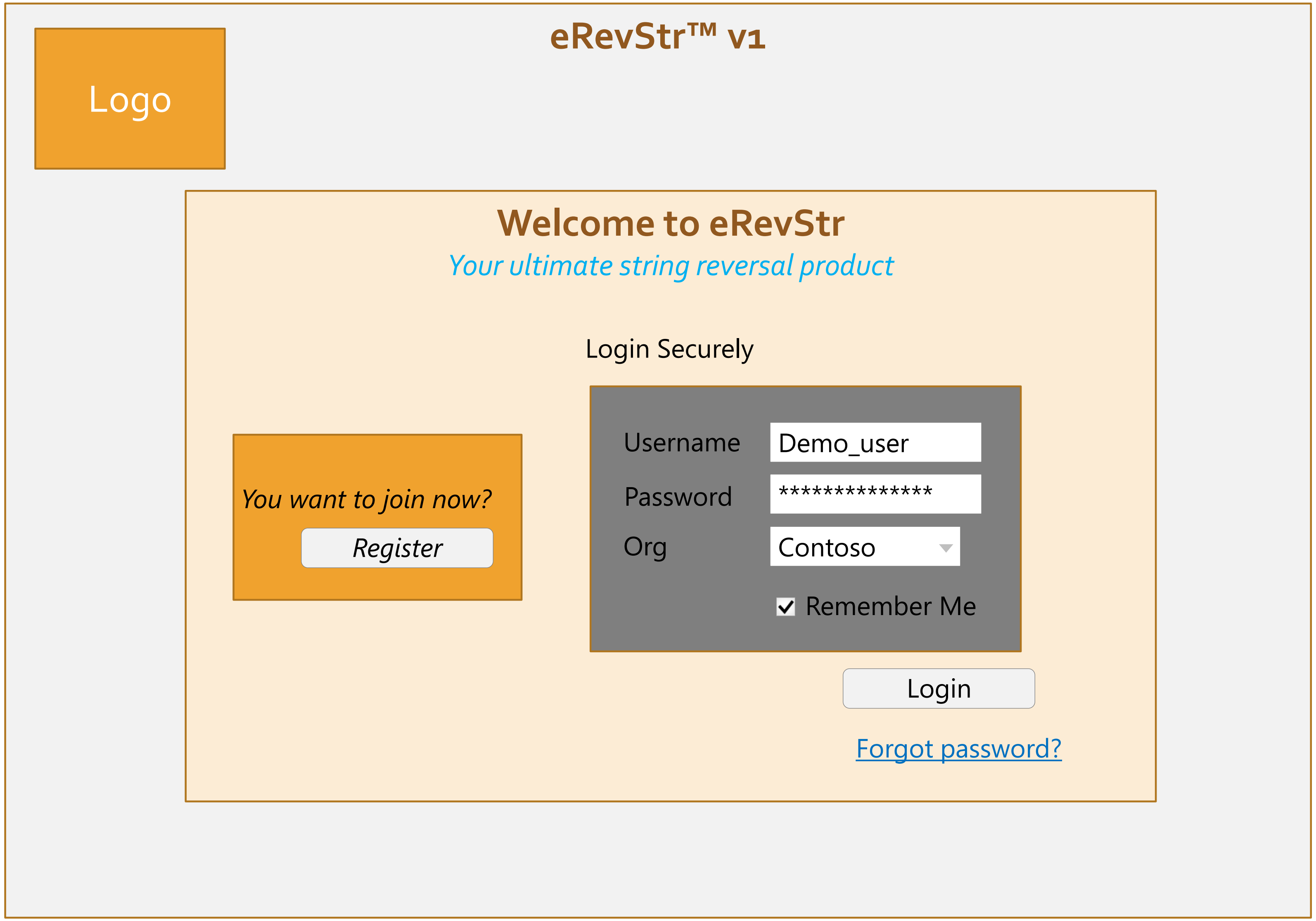
Manage Users

View/Edit
Disable /Deactivate

Conceptual UI - Login

contoso.erestr.com

erestr.com/login.aspx?id=23



eRevStr™ v1

Welcome to eRevStr

Your ultimate string reversal product

Login Securely

You want to join now?

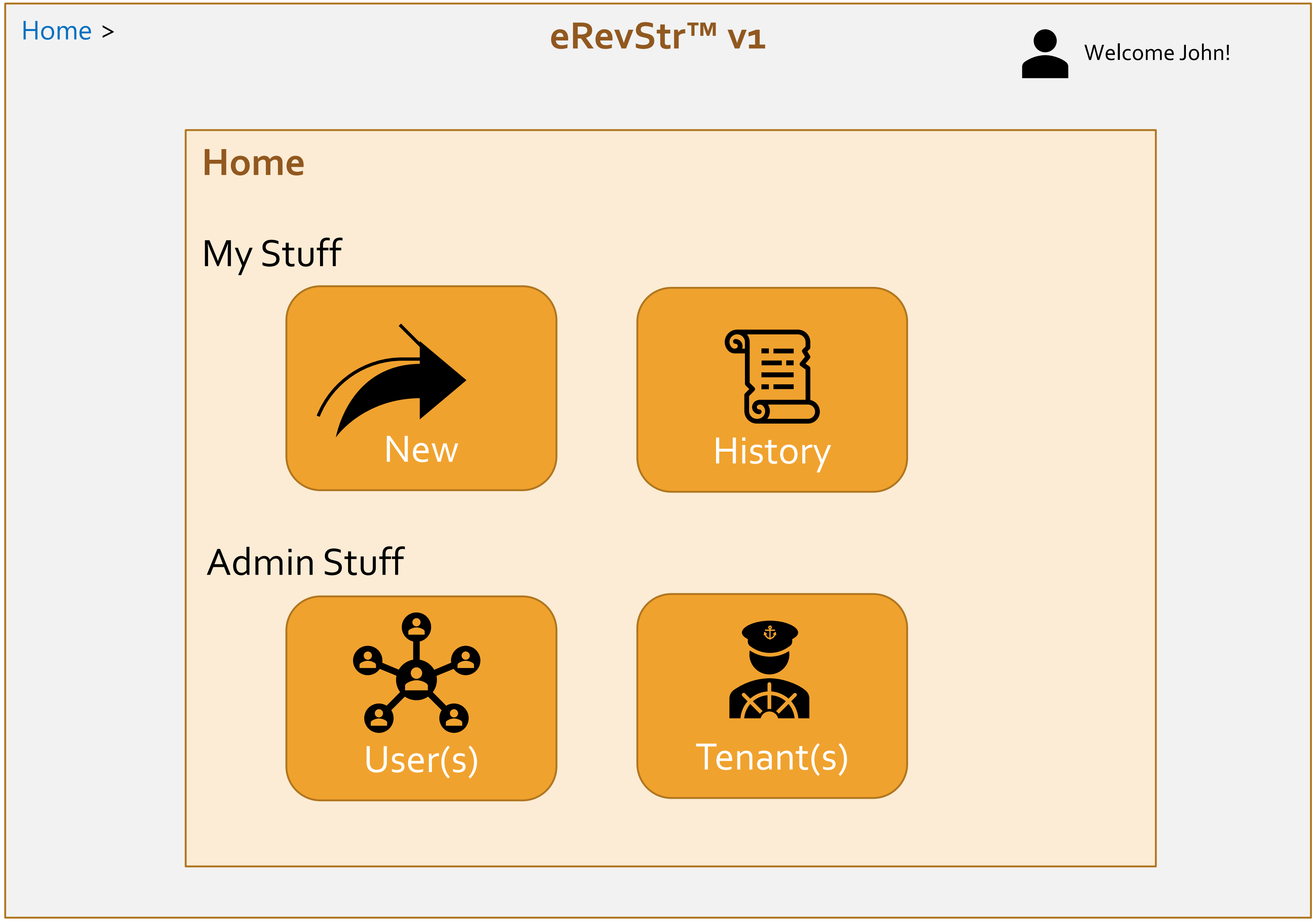
Username: Demo_user

Password: *************

Org: Contoso

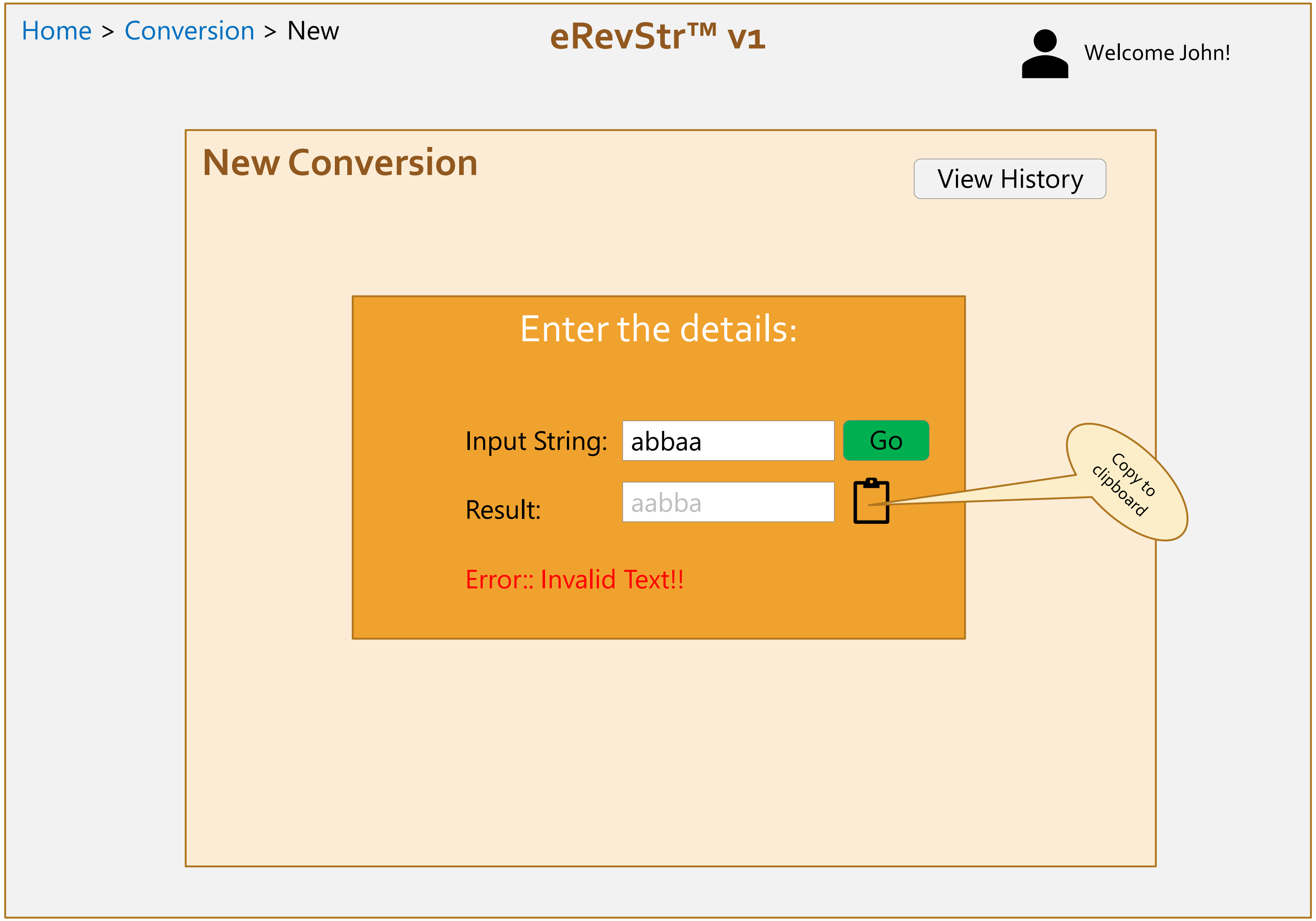
☑ Remember Me

Conceptual UI - Dashboard



Icons will be displayed based on role

Conceptual UI - Conversion



Home > Conversion > New

eRevStr™ v1

Welcome John!

New Conversion

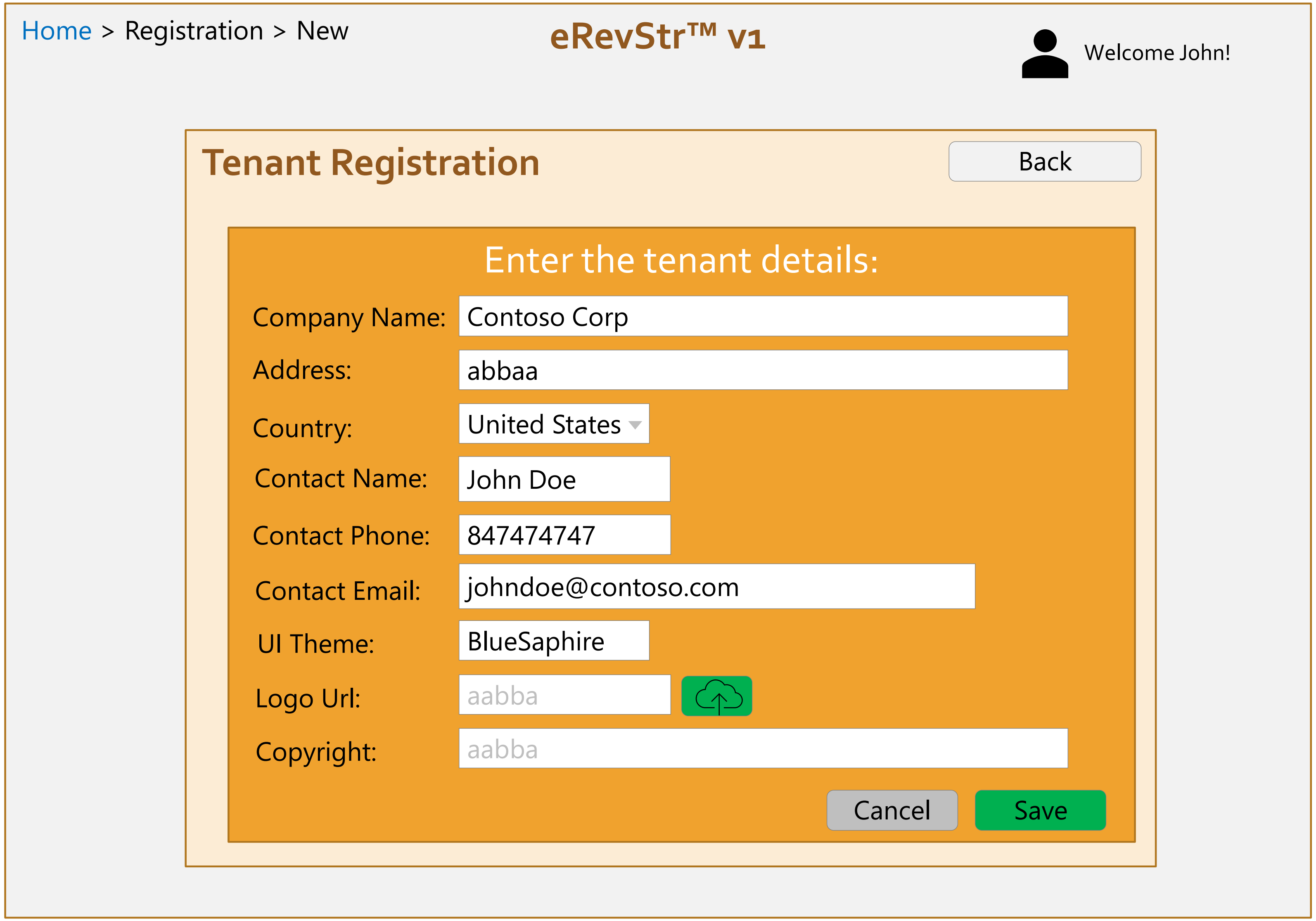
Enter the details:

Input String: abbaa

Result: aabba

Error:: Invalid Text!!

Conceptual UI - Tenant Registration



Tenant Registration

Enter the tenant details:

Company Name: Contoso Corp

Address: abbaa

Country: United States

Contact Name: John Doe

Contact Phone: 847474747

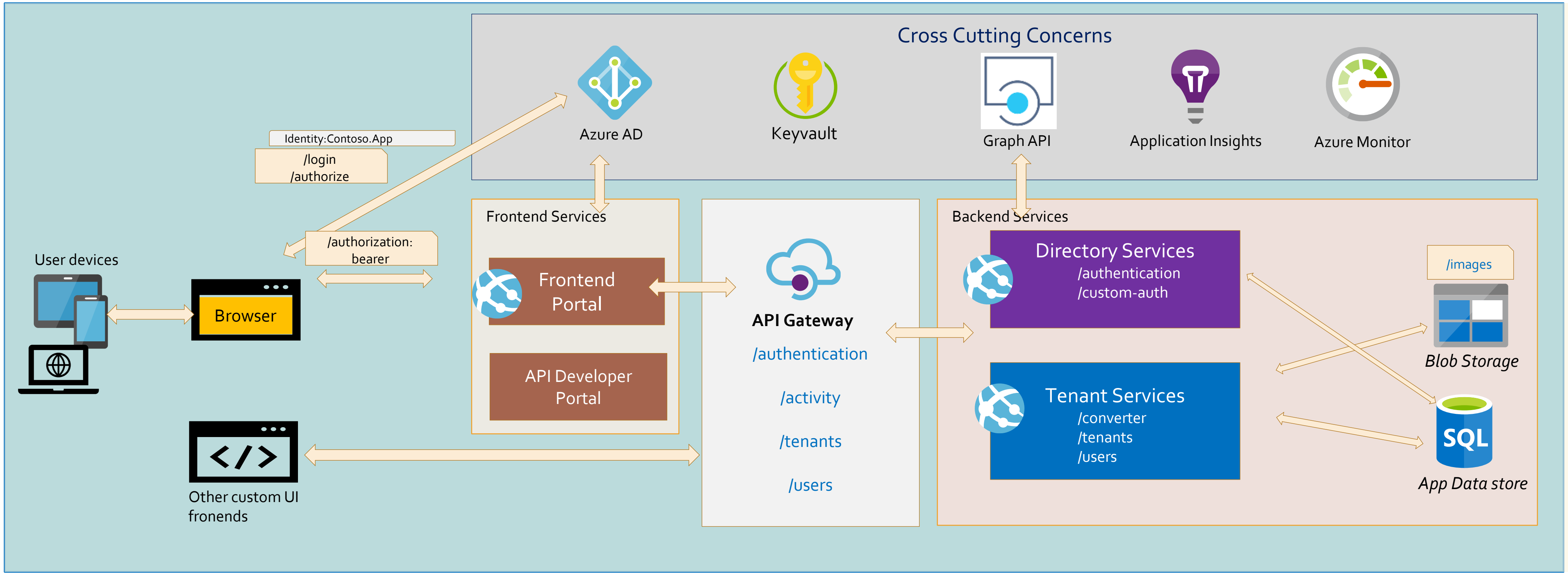
Contact Email: johndoe@contoso.com

UI Theme: BlueSaphire

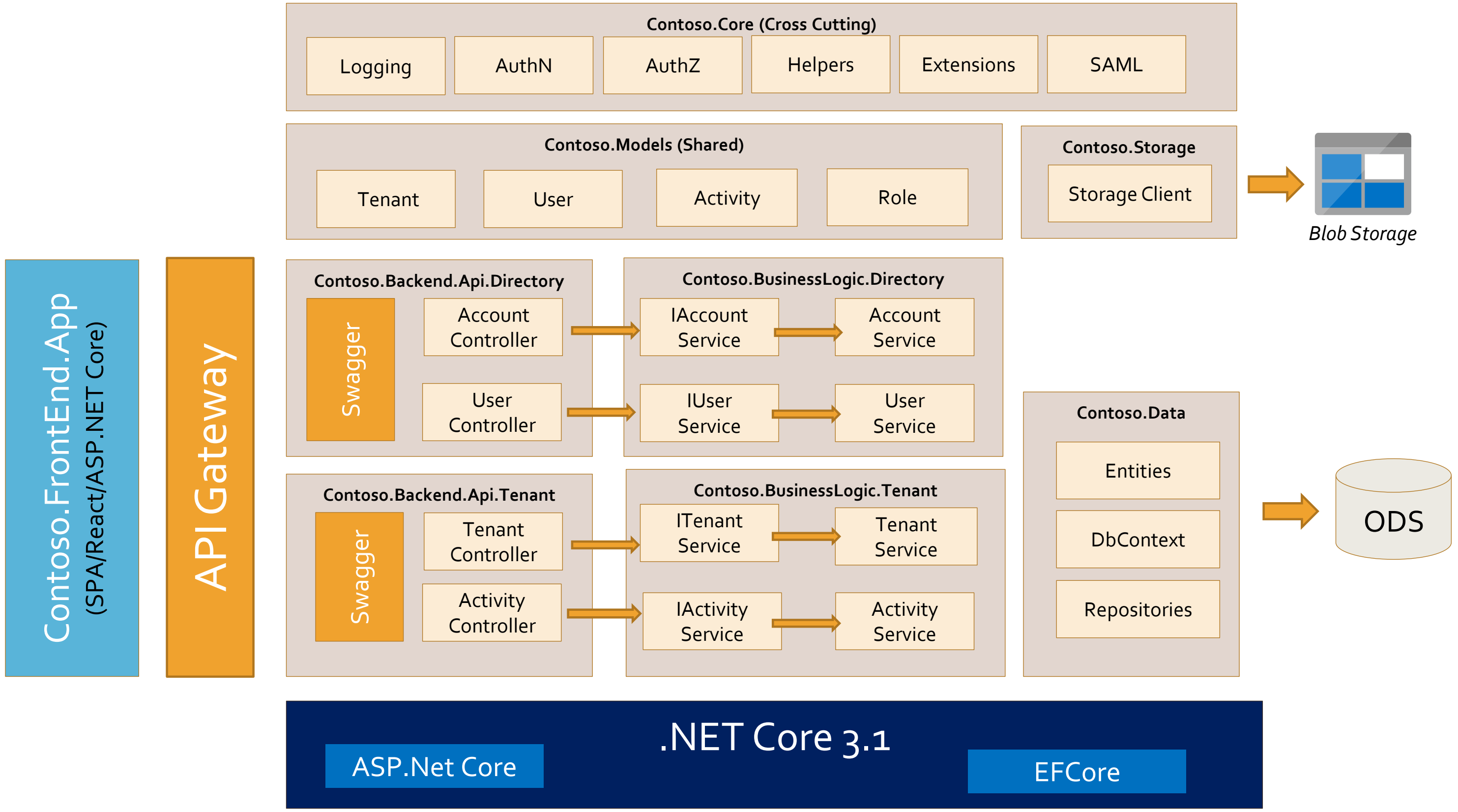
Logo Url: aabba

Copyright: aabba

High Level Architecture

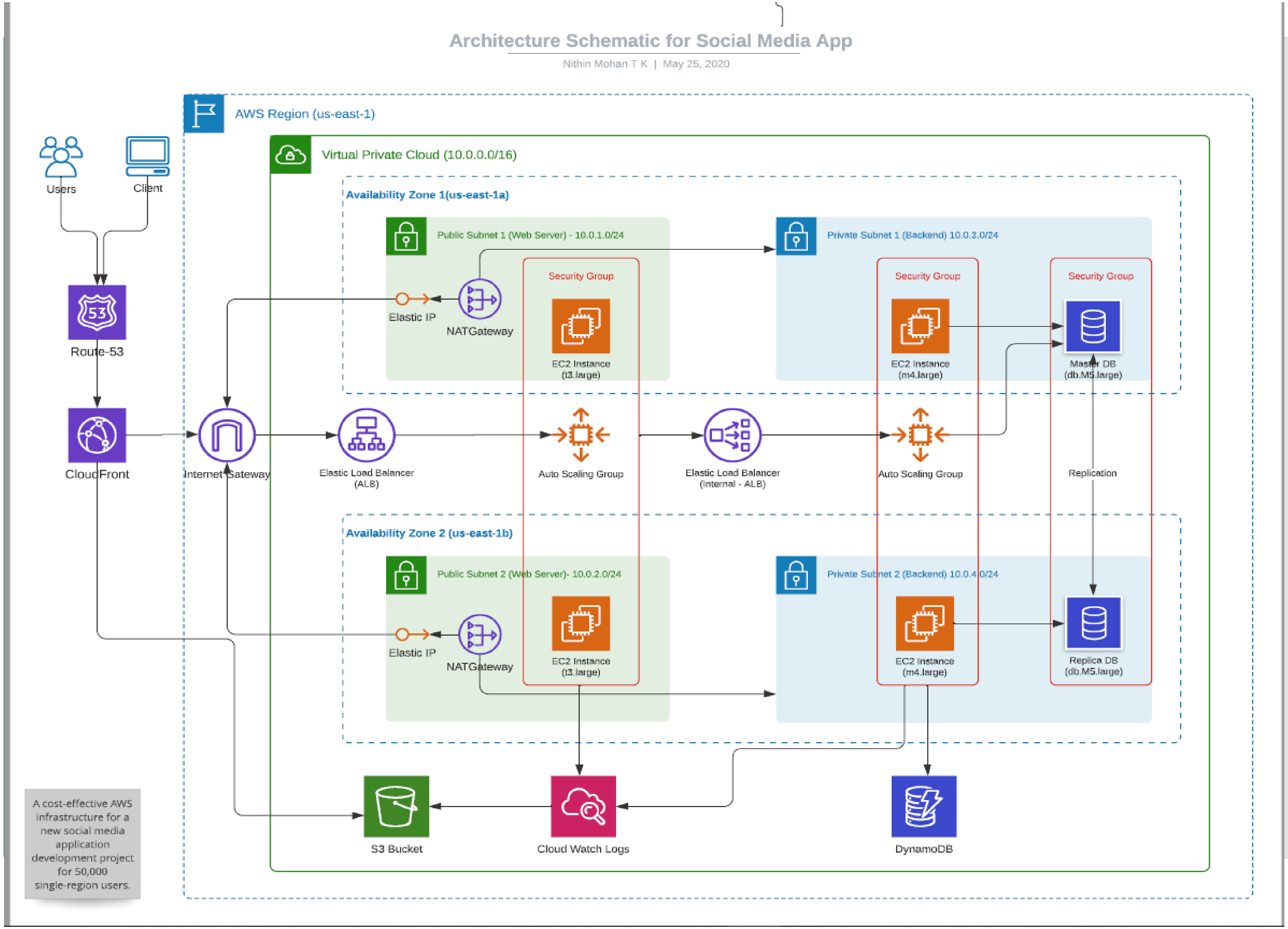


Technical Architecture



NB: *To save time to draw a highly available IT architecture, due to my current engagements – referring to one of my previous reference architecture for AWS based solution created for a Udacity nanodegree. Similar schematics would work for highly available solution in Azure.*

Infrastructure Architecture



Architecture Schematic for Social Media App

Nithin Mohan T K | May 25, 2020

A cost-effective AWS infrastructure for a new social media application development project for 50,000 single-region users.

Architecture Summary

- Architecture is with assumption that Azure AD authentication would used, and solution would use Azure as the public service provider for hosting.
- PaaS components used to ease the manageability, and scalability.
- **Frontend** would be a responsive progressive web app.
  - API provides easy replacement of frontend with Custom frontends by partners or third-party.
  - API Developer portal would act as the API self servicing portal for API. Third parties can build their custom frontend using the API documentation.
- **Scalability:**
  - System will support increasing demand of concurrent user access as user base increases.
  - CPU Metric based Autoscaling in Azure App Service would scale-out and scale-in based on the increase in demand.
- **Resilience & Availability:**
  - Azure App Service multi-region deployments would provide resilience and high availability.
  - Deployment slots would help in Hot Swap b/w deployment slots.
  - Azure SQL replica sets would ensure data redundancy and high availability.
- **Microservices & Future Cloud Native**
  - Services are built in Microservices architecture but using shared database.
  - Services can be easily decoupled with minimal effort, to be in consistent with Microservices best practices.

Security

- Default Authentication:
  - Azure AD – B2B
- Default Authorization:
  - OAuth2
- SAML 2.0 /SSO – Will support
  - PingFederate
  - Siteminder
- API Authorization:
  - OAuth2
- Secret Handling:
  - Azure KeyVault

Technology

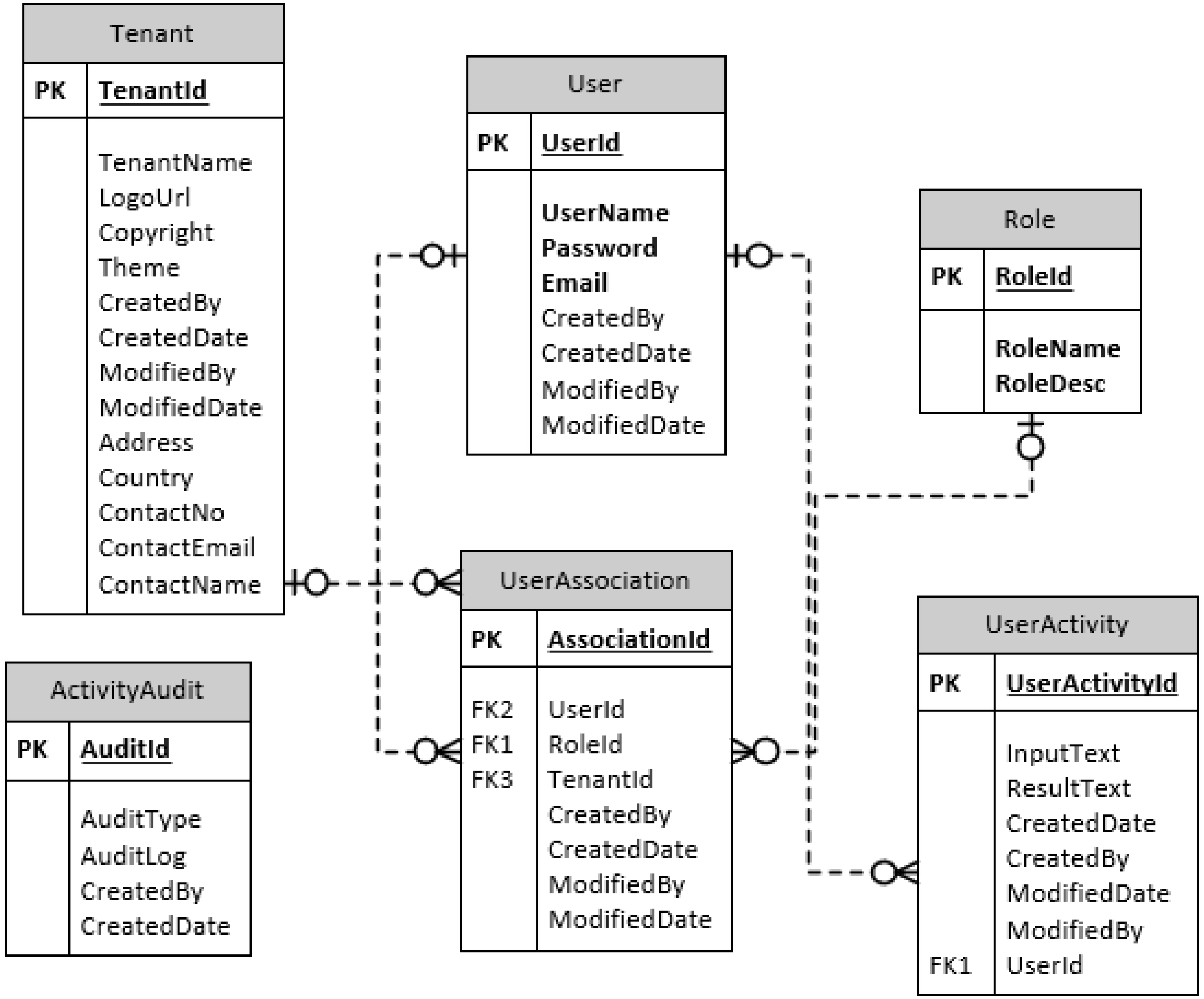
Tool Stack:

- .NET Core 3.1 / Open API
- Visual Studio 2019
- Entity Framework Core
- Swagger/OpenAPI
- Unit Testing:
    - Nlog
    - xUnit
    - NSubstitute
- Frontend
    - NextJS
    - React
    - A responsive progressive web app

Hosting Environment: Azure

- Mostly all PaaS components
- Application Insights – Telemetry
- App Service – Hosting
- SQL Server – Data Store
- Azure Storage – BLOB – Images and other files
- Api Management – api gatewayproxy

DB Schema Design



Tenant	
PK	**TenantId**
	TenantName LogoUrl Copyright Theme CreatedBy CreatedDate ModifiedBy ModifiedDate Address Country ContactNo ContactEmail ContactName

User	
PK	**UserId**
	UserName **Password** **Email** CreatedBy CreatedDate ModifiedBy ModifiedDate

Role	
PK	**RoleId**
	RoleName **RoleDesc**

ActivityAudit	
PK	**AuditId**
	AuditType AuditLog CreatedBy CreatedDate

UserAssociation	
PK	**AssociationId**
FK2	UserId
FK1	RoleId
FK3	TenantId
	CreatedBy CreatedDate ModifiedBy ModifiedDate

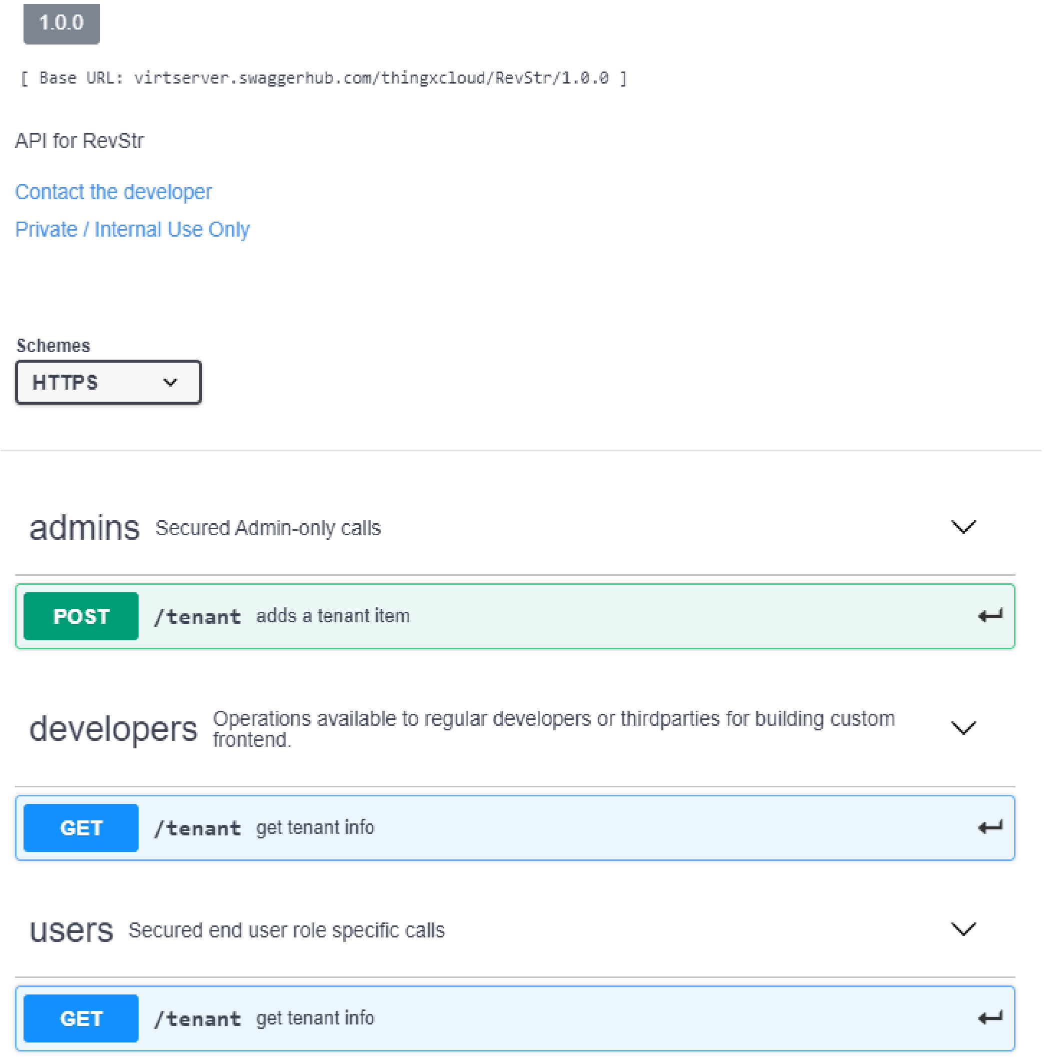
UserActivity	
PK	**UserActivityId**
	InputText ResultText CreatedDate CreatedBy ModifiedDate ModifiedBy
FK1	UserId

NB: Additionally we can add billing/payment related entities to monetize

***NB:** A mode on how this can be organized high level in swagger spec. All methods are not defined in swagger spec.*

API Definitions

- Security
  - /User/Login
  - /User/Create
  - /User/ChangePassword
  - /User/ForgotPassword
  - /User/Update
- Tenant
  - /Tenant/List
  - /Tenant/Get/{Id}
  - /Tenant/Create
  - /Tenant/Update/{Id}
  - /Tenant/Remote/{Id}
- Activity
  - /Activity/Reverse/{instr}
  - /Activity/List
  - /Activity/Get/{Id}
  - /Activity/Update/{Id}



1.0.0

[Base URL: virtserver.swaggerhub.com/thingxcloud/RevStr/1.0.0]

API for RevStr

Contact the developer

Private / Internal Use Only

Schemes: HTTPS

admins Secured Admin-only calls

POST /tenant adds a tenant item

developers Operations available to regular developers or thirdparties for building custom frontend.

GET /tenant get tenant info

users Secured end user role specific calls

GET /tenant get tenant info

Resource Requirements

Resource + Role	FTE
Scrum Master	0.5 (50% availability)
Tech Lead / Technical Architect	1
UI/UX Engineer	1
Backend Engineer (1x senior, 1x mid)	2
Frontend Engineer	1
Quality Engineer	2
DevOps Engineer	1
TOTAL Resources	**8.5**

Risks and Assumptions

- Risks
  - Availability of skilled API engineer with knowledge on OpenAPI by the project start date.
  - Performance testing needs to be conducted to ensure system can sustain the desired concurrent set of users.
  - Lack of automation can result in escaped defects.
  - Post-launch plan for marketing the product require early involvement of Marketing team.
- Assumptions
  - Solution is built with assumption that Azure AD authentication would used, and solution would use Azure as the public service provider for hosting.
  - Default language supported by the portal will be English.

Future Scope

- Microservices can be containerized easily as we choose DotNetCore 3.1 as the underlying runtime platform.
- Database can be split in to dedicated self sufficient individual database to provide individual scale or to enable individual components can be deployed independently as they change.
- We can use Azure Container Service or Kubernetes for orchestration.
- Additional security and trust can be enforced with certificate based authentication b/w Client and Server.
- Can support on-premise deployments with proper installation documents and licensing integration.

Additional Thoughts

- Use Functions for Compute – string reversal process can be offloaded to a function, as it can scale up to number of instances and we only need to pay for execution.