**HAPPIEST MINDS TECHNOLOGIES**

**PropertyGuru Section B – Technical Response**



**Copyright Information**

This document is the exclusive property of Happiest Minds Technologies Ltd. (“Happiest Minds”). The recipient agrees that they may not copy, transmit, use, or disclose the confidential and proprietary information in this document by any means without the expressed and written consent of Happiest Minds. By accepting a copy, the recipient agrees to adhere to these conditions to the confidentiality of Happiest Minds practices and procedures.

**Confidentiality Clause**

This document is being submitted to PropertyGuru by Happiest Minds Technologies Ltd. on the understanding that the contents of this document will not be divulged to any third party without the express written consent of the parties. It is also understood that the parties will not divulge any confidential information about PropertyGuru that it may have access to during this interaction.

**Disclaimer**

This document has been prepared based on the information provided by PropertyGuru . Wherever proposed, the solutions and/or services mentioned are based on the requirements defined and understood by us at the time of preparing this document. While every effort has been made to make this document as accurate as possible, there might be changes to the document based on the subsequent discussions.

[**1.0** **Executive Summary 5**](#_Toc133084189)

[**2.0** **Overview 6**](#_Toc133084190)

[**3.0** **Requirement Details 6**](#_Toc133084191)

[*3.1* *Functional Requirements 6*](#_Toc133084192)

[*3.2* *Non-Functional Requirements 16*](#_Toc133084193)

[*3.3* *In Scope 17*](#_Toc133084194)

[*3.4* *Out of Scope 18*](#_Toc133084195)

[*3.5* *Assumptions 18*](#_Toc133084196)

[*3.6* *Dependencies on PropertyGuru 19*](#_Toc133084197)

[**4.0** **Solution Design And Approach 19**](#_Toc133084198)

[*4.1* *Design considerations 19*](#_Toc133084199)

[*4.2* *Modernization Strategy 21*](#_Toc133084200)

[*4.3* *Layered Architecture 25*](#_Toc133084201)

[*4.4* *Technical Risks and Mitigation 26*](#_Toc133084202)

[**5.0** **Testing and Validation 27**](#_Toc133084203)

[*5.1* *Baselining of Artifacts 27*](#_Toc133084204)

[*5.2* *Test Requirement Analysis 27*](#_Toc133084205)

[*5.3* *User Scenario Analysis 28*](#_Toc133084206)

[*5.4* *Test Cases Design 28*](#_Toc133084207)

[*5.5* *Type of testing 29*](#_Toc133084208)

[*5.6* *API Automation 35*](#_Toc133084209)

[*5.7* *Test Types Vs Test Levels & Environments 36*](#_Toc133084210)

[*5.8* *Test Execution & Defect Reporting 36*](#_Toc133084211)

[**6.0** **DevOps Strategy 37**](#_Toc133084212)

[*6.1* *Deployment Architecture for EKS 37*](#_Toc133084213)

[*6.2* *AWS Governance 40*](#_Toc133084214)

[*6.3* *DevOps CI/CD Pipeline 41*](#_Toc133084215)

[*6.4* *Strangler DevOps Pipeline 43*](#_Toc133084216)

[*6.5* *Branching strategy 43*](#_Toc133084217)

[*6.6* *Release Management 45*](#_Toc133084218)

[*6.7* *Disaster Recovery 46*](#_Toc133084219)

[*6.8* *Proposed Approach for Disaster Recovery 47*](#_Toc133084220)

[**7.0** **Technology Stack – Indicative 48**](#_Toc133084221)

[**8.0** **Warranty and Maintenance Support 48**](#_Toc133084222)

[*8.1* *Scope and Requirements 48*](#_Toc133084223)

[*8.2* *Support requirement 48*](#_Toc133084224)

[*8.3* *Out of Scope 49*](#_Toc133084225)

[*8.4* *Definition of Priority 49*](#_Toc133084226)

[*8.5* *Solution consideration 49*](#_Toc133084227)

[*8.6* *Post Implementation – Support Operating Model 50*](#_Toc133084228)

[*8.7* *On Going Support Task details 50*](#_Toc133084229)

[*8.8* *Timeline 50*](#_Toc133084230)

[*8.9* *Major Incident Management 51*](#_Toc133084231)

[*8.10* *Assumptions 51*](#_Toc133084232)

[*8.11* *Indicative SLAs, Response and Resolution Times 52*](#_Toc133084233)

[**9.0** **Schedule and Milestones 52**](#_Toc133084234)

1. **Executive Summary**

Happiest Minds is pleased to present this response to PropertyGuru (PG) for building the Regional Marketplace Platform project aimed to migrating the monolithic Guruland platform to microservices based architecture. In this response to the RFP, we have attempted to address the various facets that PropertyGuru is looking for in a partner and have also provided some additional information that should allow for an informed comparison and decision. Happiest Minds, while being a relatively young company, has managed to emulate the revenue and achievements of much larger companies to establish a fiscally sound platform for sustained growth. The company has its base in Bangalore India, which allows it to access one of the best software talent available globally across a wide range of technologies in a highly scalable manner.

The company’s founding rationale was focus on digital technologies such as Cloud, Mobility, Big Data & Analytics, Social Computing, Security and IoT which when superimposed lead to our specialization in the Digital Transformation that many companies are undertaking. Recognizing the differences and complexities involved in Product Engineering Services as compared to traditional IT services targeted at enterprises, Happiest Minds established an independent Business Unit to focus on platform and product companies and build a team of product engineering specialists to cater to these companies. We propose to constitute the team for PropertyGuru from this experienced pool of product engineering talent. The Product Engineering Services BU at Happiest Minds has worked extensively with both B2B and B2C companies helping build and launch various products & platforms and we will distil out learnings and best practices from these engagements to add value to this engagement.

As part of our response the following documents are being submitted to PropertyGuru

|  |  |
| --- | --- |
| SectionA\_HappiestMinds\_22APR\_Version1.0.pdf | This document provides an overview of Happiest Minds and its capabilities including the relevant offerings and capabilities for PropertyGuru |
| SectionB\_HappiestMinds\_22APR\_Version1.0.pdf | This document lays down our understanding of the requirements, assumptions and our technical approach. The salient point of the technical approach is to emphasize “Business as Usual**”**, which is why we have suggested a Strangler Pattern approach for implementing microservices |
| SectionC\_HappiestMinds\_22APR\_Version1.0.pdf | This document provides sample profiles of the key roles envisioned for the project |
| SectionD\_HappiestMinds\_22APR\_Version1.0.pdf | This document explains the delivery methodology, governance, and communication mechanism |
| SectionE\_HappiestMinds\_22APR\_Version1.0.pdf | This document details out a few success stories with reference to migration legacy systems, microservice based implementation, eCommerce implementations |
| SectionF\_HappiestMinds\_22APR\_Version1.0.pdf | This document outlines our commercial proposition. This quote is subject to revision after the discovery phase based on the additional information/knowledge acquired during Pjase-2 discovery |
| SectionG\_HappiestMinds\_22APR\_Version1.0.pdf | This document lists additional Terms and Conditions |

We are confident that Happiest Minds would be an ideal partner for PropertyGuru given our focus on leveraging the latest technology innovation to help organizations transform their business.

1. Product Engineering DNA – Deep expertise in creating product and platforms
2. Extensive Experience with Modernizing Legacy systems
3. Transparency in terms of risks, milestones and progress
4. Metrics driven delivery process to ensure robust governance and project execution
5. Trusted Advisors to our customers - Focused on providing rich industry specific consulting services to create the roadmap for success

We sincerely hope this proposal is in line with PropertyGuru’s expectations, and we look forward to this partnership.

1. **Overview**

PropertyGuru is Southeast Asia's #1 digital property marketplace, with leading positions in Singapore, Malaysia, Vietnam, and Thailand. PropertyGuru currently hosts more than 3.5 million real estate listings, 40 million property seekers monthly with more than 64,000 active real estate agents across five growing economies of Southeast Asia. In 2021, PropertyGuru completed the acquisition of iProperty Malaysia and ThinkofLiving.

PropertyGuru has a single monolithic application called Guruland. This code serves all its marketplaces in Singapore, Thailand, Malaysia, and Indonesia. Guruland software was built with technologies that were in vogue in the market at that time. Over time with expansion of business, new features were continuously developed and added to this monolithic codebase. PropertyGuru intends to revamp its legacy system with a suite of small and autonomous microservices. The project named “Regional Market Platform” is a concerted effort in the direction of rolling out the legacy system in a phased manner.

1. **Requirement Details**
   1. **Functional Requirements**

List of microservices identified for migration to new architecture.

|  |  |
| --- | --- |
| **Service** | **Functions** |
| Listing Management Service | Build a new CRUD-based listing management service with a purpose-built data store, thereby omitting the dependency on PropertyDB.   * Post   + Select the tenant for which a Listing is being created   + Define the Listing details (type of Listing, Listing Images, description, tenure, price range, valid until, landmark, size, add Agent Details etc.)   + Create tags for Listing - Like Built2018, Freehold * Get   + View All the created listing (Active/ Expired)   + Paginated List, Search (Listing ID), and Filter (date range, category, location, price range)   + Retrieve listing performance   + Retrieve listing rank   + List quality score   + Listing creates date   + Listing expiry date   + Listing reposts date   + Listing Views   + Listing impressions   + Listing Suspension - View suspension details, Appeal suspends listings, View Previous comments, Add comments   + Retrieve Performance report   + List property types   + List Boosted bookings   + List Past listings * Update   + Update any listing   + Update the attributes of a Listing like (Image, contact details, property description, and tenure etc.)   + Repost listings - View Ad credits, Schedule Auto Repost listing   + Update listing status - mark rent/sold   + Reactivate suspend listings   + Update report listing rule * Delete   + Delete/Deactivate/Expire any Listing * Check PropertyGuru Suggested Price   + Check price Suggestions from PropertyGuru   + At click of this, PropertyGuru suggested Price for the property should be displayed. The price should be as per the defined currency for the website   + This helps in quoting the price for the listed Property. * Toggle Feature   + Some functionalities are managed using feature-toggle, e.g. Tenant Based configuration (Make it live on selected site), Agent-Based configuration (don’t show to specific agent group) * Content Validator   + To filter out unwanted language (racist keywords) from the listing details, content validator is used * Media Service   + Media related to listing is being delivered by the Media service (images, videos, floorplans etc.) * Listing Performance   + Analytics around a specific listing is being handled by this service.   + Listing performance overview   + Listing performance insights   + Listing competition summary   + Listing price insights * Listing Alert   + See when a New Listing is Posted, Deleted, Updated, Shortlisted * Listing statistics   + View listings data by date range and up to past 365 days * Shortlist listing   + See all the Shortlisted Listings [Wishlist kind of feature] * Recommendation   + Retrieve recommendation listing |
| Listing Search Service | Considering the difference in PropertyGuru, iPP and BDS revamp this service to consolidate the search-related APIs to this unified service, including building a new API to pull expired listings from the master listing database. The main objective of having this service is to have the separation of concern between agent and consumer business.   * Search Listings based on role type.   + Show Expired/ Live Listing / both.   + Listing Result Details - Less info vs More info * Listing Search Use cases   + Suggestion based search.   + Search based on location (City, district, area)   + Search based on house Type   + Search based on Category Type [ Buy, Sell, Rent]   + Null Search   + Search based on Listing Attributes [No image search or voice search]   + Search based on Listing tag   + Search by MRT [train Stations]   + Search by HDB Estate   + Search by some Parameter   + Last searched Timestamp to be maintained   + Search based on type of transaction (residential, commercial, land, agriculture) * List Search Result   + View Listing on Listing Result Page   + View Listing on Google Map   + Filter and Sorting options - Filter Listings by Recommended, Newest, Price, size)   + Save a Search result - Need login to save   + Display Agent Profile and Contact information with listing   + YouTube link for search Results   + Send Email to the Agent; opens in outlook (default)   + WhatsApp Integration with search result   + Hide or unhide any Listing   + Option to See and give Agent Reviews |
| Agent Subscription Service | A multi-tenant service which will host all the functionalities related to agent subscription, i.e., credits management, billing of agent products, and self-serve payments. It should support agent subscription-related workflows.   * Buy a package.   + List packages - Standard, Advance, Premier, Business   + Accept/Reject quote   + List payment types - Full payment, pay-per-month, pay-quarterly, pay-half-yearly   + Make Payment - Third party payment integration * Renew a package   + List purchased packages   + List Package expiration date   + List quote   + Accept/Reject quote   + Make Payment - Third party payment integration * Upgrade a package   + List available upgrade packages   + List quote   + Accept/Reject quote   + Make Payment - Third party payment integration * Top Up Prime Credits   + Buy Prime Credits   + Make Payment - Third party payment integration * Top Up Ad credits   + Buy Ad Credits   + Make Payment - Third party payment integration * PropertyGuru Money   + List Ad credits balance, expiry date   + Retrieve Ad Credits Consumption Report with filters   + Partner 360 Rewards tier   + List Package subscription   + List total number of Prime Credits   + List total number of Ad Credits * Ad Credits Spend Summary   + Views per Ad credit   + Leads per Ad credit   + Impressions per Ad credit   + Ad Credits Overview   + Itemized Ad spendings detail |
| Agent Management Service | A multi-tenant Agent Management Service that handles Agent domain for PropertyGuru marketplaces. It is expected to be created as a standalone microservice which will own the Agent and Agency domains. It will provide the APIs for the Agent UI service, i.e., acting as a backend for the frontend (“BFF”).   * Manage Agent   + Create Agent records   + List Agents   + Search Agents   + Filter Agents   + Update Agent records   + Send/ Resend Verification Email to Agent   + Send/Resend Activation SMS   + Retrieve Agent delayed membership.   + Retrieve last 30 days stats of an Agent   + Suspend Agents   + Delete Agent record.   + Sync user info to third party * Manage Agency   + Create Agency record   + Update Agency record   + Search Agency * Manage Company   + Update company   + List company by ID * Manage featured agents   + Search featured agents   + Expire featured agents   + List Featured agents * Manage Agent's Suspension   + List agent's suspension records   + Update appeal request   + Delete appeal request * Manage Users   + Create User   + Search Users   + Filter Users   + List Users * Manage User address   + Create User address   + Update user address   + Search user address records   + Delete a user address   + List user address * Manage Transactions   + Creates transaction record   + Update transaction record   + Search transactions   + List transactions * Manage role based access   + Create access rules   + Update access rules   + Delete access rules   + List access rule * Manage communication log   + Create communication log   + Update communication log   + Search communication log   + Delete communication log |
| Users Service | The user service will be a completely standalone service which owns the User domain.   * Remove any dependency that this service has with legacy PropertyDB tables * Registration   + Role: Admin:   + CRUD ACL Rules   + Role: Agent/ Consumer   + Register as an individual or Company. * Communication preferences   + Role: Consumers/ Agents / Admin   + CRUD Communication Preferences [ EMAIL, SMS, WhatsApp] * Session management   + Configure Auto Sign-out time   + Send notification when logged in from another device   + Manage Session logs |
| Mobile BFF | A separate BFF layer for mobile devices common across all PropertyGuru marketplaces with exhaustive APIs which can power PropertyGuru mobile apps   * Property Agents API   + Sync user info   + Search agents   + Search by Agent id * Agent-website   + Update agent domain expiry date * Alert API   + Create alert.   + Search for alerts   + Alert count   + Delete alert   + Retrieve alerts * Autocomplete search   + Autocomplete search for Property Listings * Autocomplete-agent   + Autocomplete search for Agent records * Boost-bookings   + Retrieve booked bookings (active and delay) by listing id   + Retrieve booking configuration * Mortgage calculator   + Saves calculated loan eligibility criteria of customer   + Invoke mortgage calculator widget * Developers   + Create developer account * Dfp-targeting   + Google DFP targeting * Discretionary Product API   + Retrieve Agent specialists * Enquiry   + Create new enquiry for agent   + Create new enquiry record for the listing   + [Developer] Create new enquiry record for listing * Property Listing API   + Create new listing   + Search for listings   + List banned words/expressions   + Listing presence and validation rules   + Retrieve single listing   + Update listing with given ID   + Reactive expired listing - deduct credits   + Resume listing - Activate listing   + Update listing with given ID * Billing API   + Update start date of Agent * Map   + Map search for cluster of listings * Media API   + Retrieve image   + Upload an image   + Delete a media item   + Edit media attributes * Media – moderation   + Add media to moderation queue   + Retrieve moderation status of set of media   + Retrieve moderations status of set of listings   + Add moderation appeal for media * Package   + List packages for PropertyGuru Money   + List summary of packages for PropertyGuru Money * Price Index   + Search for price index * Products   + Create product * Properties   + Search for properties   + Price index for the listing/property   + Price index for District   + Search for property   + Search POI around property   + Price index for the Property   + Price index for property valuation   + Transaction for property * Purchased products   + Retrieve purchased count and utilization by user   + Update purchased product   + Activate purchased product   + Search purchased products by filters   + Expire purchased product by ID * Purchases   + Create Purchase   + Utilized Ad credits products * Reference   + Geographic reference data   + Reference data for an area/district   + Retrieve districts in a region.   + Retrieve reference data for an area   + Retrieve Listing related reference data * Report   + Ad-credits reports   + Total Ad-credits report * Shortlists   + Add items to shortlist.   + Search for shortlist items   + Delete shortlist item * Transactions   + Rentals and Sales data   + Rental and Sales data trends * User   + Signup a user -Initiates sign up then redirects to PropertyGuru web * Utilized products.   + Create a utilized product   + Deletes a utilized product * News API   + Retrieve news details based on parameters   + List all categories in a country   + Search News based on parameters   + Search News   + Retrieve list of news based on one or multiple tags * New Project reviews   + List reviews by language and region   + List reviews   + Terms Review * World data   + POI train stations from world SDK   + MRT stations * Property Developer   + Search for developer listings   + Count of developer listings   + List of featured listings by country   + Last video   + List videos   + Details of specific video   + New Home listing details   + List of Featured listings   + Search for overseas listings   + Count of world developer listings   + List of overseas features listings   + List of overseas new home listings |
| Admin Service (Frontend) | Create a micro-service which will cater to all the admin-related tasks such as impersonating an agent, managing agents, agencies, users, and other domain objects. In essence, it will be a single place from where we would be able to manipulate all the marketplaces within PropertyGuru. We will also try to find out if the missing features from Salesforce could be added to this new service.   * Admin Service   + Act as Superuser to manage other users' and agents' accounts * Impersonating an agent   + Search for any Agent and look at the agent's profile, listing, and other details and use its account to perform functions like posting ads, replying to end customer etc. * Manage agents   + Active/ Inactive an Agent   + View Agent Profile Details   + Filter, search * Manage Agencies   + Active/ Inactive Agencies   + View all the agents under an agency.   + Filter, search * Manage Users   + Active/ Inactive Users   + View User Profile Details   + Filter, Search |
| User Engagement – Enquiry | A multi-tenant service to handle all the leads and enquiries originating from any of the marketplaces. As part of the revamp, the dependency on legacy PropertyDB needs to be removed, and a purpose-built database for this service is to be created. It will also make use of a messenger service to send different types of communications, i.e., email, SMS etc.   * Comment   + Send Comment for any selected Listing   + See list of Comments and Responses [ Reference LinkedIn/ Teams kind of chat where all the chats are listed on the left]   + Delete a Comment   + Update a Comment [edit] * Content Validator   + Ability to Report an offensive Content.   + Get banned words/expressions * Enquiry   + Send enquiry for any listing - Share Email ID, Phone number, and Name   + Enquiry Creates a Lead in Salesforce   + All enquiries go to an inbox [ under My Activity]   + The agent responds to the Enquiry * Feedback   + Provide Site Feedback   + Give your - Name, email , mobile, text and CAPTCHA |
| Advertising | A multi-tenant microservice to handle use cases related to featured agents or listings for all PropertyGuru marketplaces. To create this as a standalone service, would need to remove the dependency from the legacy PropertyDB and migrate the tables to the purpose-built DB for this service.   * Featured Agents(CRUD):   + Featured Agent category   + Book Featured Agent slots   + Manage Featured Agent slots   + Extend booking   + Notify when slots available   + Reserve slot   + Set duration   + Schedule start/end date * Promoted Listings:   + List Active/Inactive/Sale/Rent promoted listings   + Promoted Listing Metrics   + Get Advertising campaign metrics   + Begin/End campaign   + Set priority * Advertising:   + Select Tenant(s)   + Option to create Advertisement for Agent/ Listing   + Option to Pull video/ image from media services   + Select the real estate [ Left Banner, Right banner, hero banner] and Name of the Page [ Listing Page, Detail Page]   + Define the schedule of the Advertisement going live   + Ability to Cancel/ Stop or edit and update any advertisement before end time   + View Past Advertisement and filter by Category/tenant/date   + use old advertisement template and make edits to create a new Ad.   + Use Ad Credits for posting advertisement * Custom Categories for advertisement   + Create Custom Categories for Advertisement [ On Sale, Easter Deals, Listings With good reviews, bad inventory etc.]   + Select the tenant and Schedule the Launch time and date   + Select Listings for these categories - Search, Filter options |
| Developer (Database Dependencies) | The interaction of Project Net with PropertyDB needs to be omitted as part of this revamp. The dependent Tables need to be migrated to the respective services to ensure the services are bound to their own domain objects. This will make ProjectNet a standalone service in the true sense.   * Project Net - PropertyDB dependency   + location   + media   + media\_item   + media\_original   + property   + property\_amenity   + property\_details   + property\_status\_history   + property (view) * ProjectNet – ETL   + streetnames   + location   + property   + property\_amenity   + property\_status\_history   + property\_details   + property\_type   + property\_unit |
| Info Service | A multi-tenant standalone microservice which will be responsible for handling all the requirements related to the price index, POI, transactions, valuations, property reviews etc.   * POI   + Point of Interest * Price Index   + Demand Index   + Price Index   + Property price index   + Subzone list demand * Price Index listing   + Retrieve Price Index * Properties   + Featured properties   + Search properties using free text and filters   + Property detail list, unit detail if floor and stack provided   + Property detail ID, unit detail if floor and stack provided.   + Point of Interest near to property   + Price Index of property   + Rentals and Sales data of particular property   + Rentals and Sales data of near a particular property * Properties valuation   + Valuation of a particular property   + Information about regional transactions used for the valuation   + Information about property transactions used for the valuation   + Valuation of a particular unit * Property reviews   + List Property reviews   + Post a review for a particular property   + Details of property review   + Report a property review   + Submit a property review   + Update a property review   + Vote a property review * Territory pricing   + Local pricing information by district code   + Local pricing information by region code   + Local pricing information by territory code * Transactions   + Transaction data of sales or rentals for HDB or URA   + Rentals and Sales data   + Rentals and Sales trends data |
| Email Template Management | Move all the legacy email templates to the messenger service. The messenger service should be further enhanced to expose an interface where admins would have capabilities to manage different types of email message templates using WYSIWYG capabilities.   * View List of Email Templates   + View all the migrated multilingual templates based on region.   + Upon Selecting a particular Template, the default email tool should open (e.g., outlook) * Create a New Email   + Ability to define a New Email template Category.   + Rich Text Editor (WYSIWYG capabilities) to create a new template   + common header and footer and standard template structure for reusability & extensibility * View Past Emails   + View List of all the past Communications   + Ability to search or Filter based on category, Subject, Customer * Promotion   + Promotion and marketing emails |

* 1. **Non-Functional Requirements**

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| **Code Coverage** | 98% code coverage for the services developed in this engagement |
| **Security** | * + - * Ensure that the design and implementation of the Software shall not have any vulnerability listed under OWASP Top Ten, which includes:         + Broken Access Control         + Cryptographic Failures         + Injection/Cross-site Scripting         + Insecure Design         + Security Misconfiguration         + Vulnerable and Outdated Components         + Identification and Authentication Failures         + Software and Data Integrity Failures         + Security Logging and Monitoring Failures         + Server-Side Request Forgery         + Cross-Site Scripting (XSS)         + Broken Session Management         + Insecure Direct Object References         + Cross-Site Request Forgery (CSRF)         + Insecure Cryptographic Storage   + Failure to Restrict URL access.   + Insufficient Transport Layer Protection   + Unvalidated Redirects and Forwards   + Buffer overflow   + Improper Error Handling   + Monitor and Limit ingress traffic for anomalous behavior and suspicious activity   + Data Encryption both for transit and Rest |
| **Performance** | The performance metrics will be discussed and finalized during the discovery phase. The guideline for performance is given below:   * No degradation in performance, the minimum performance levels for the new microservices are expected to meet the existing system performance * Scalability of the system at different data loads |
| **Disaster Recovery** | The Recovery Time Objective (RTO) and Recovery Point Objective (RPO)  metrics will be discussed and finalized during the discovery phase |
| **Documentation** | * Wiki documentation for Architecture, system design, sequence diagrams, deployment diagrams, and use case diagrams * Swagger documentation for the API with details on expected input, output and edge cases. * Build and deployment details * Test Strategy and Plan * Knowledge transfer documents |

* 1. **In Scope**

1. Decomposing existing Guruland monolith codebase into multitenant microservices. Changes are limited to PropertyGuru in SG, TH, MY, and ID marketplace.
2. Breaking existing monolith Guruland database schema to smaller schema aligning with the new microservices
3. Data Migration from monolith system to microservices
4. Web and App frontend integration with newly developed microservices
5. Bifrost, Data Bridge, Salesforce, NetSuite, Workato, Segment and Braze changes limited to integration with new microservices.
6. Functional and DB Testing
7. API Performance Testing (for newly developed microservices)
8. Test Automation for the new Microservices (API automation)
9. Provide support for setup and automate the Infrastructure and CI/CD pipeline on AWS cloud for Dev, Testing, Staging and Prod environments
10. Knowledge Transition is limited to System Architecture, Database Schema and new microservices codebase.
11. Post-production 12 months warranty support (L3 Support)
12. Support Security Testing (conducted by PropertyGuru or identified 3rd party)
13. All knowledge transition and technical documentation would be in English
    1. **Out of Scope**
14. Any services not mentioned in requirement section
15. Migration or decomposition of any service or codebase outside Guruland codebase
16. iProperty (“iPP”) in Malaysia, Batdongsan (“BDS”) and thinkofliving in Vietnam Code/Service migration
17. Mobile and Web UI new feature development
18. AI/ML Services and related modules
19. Migration or changes to CMS layer, static pages including Agent offering and Area Insider applications.
20. Change in flow for 3rd party Advertisements
21. UI/UX Design
22. Post-production infrastructure support
23. L1/L2 Support
24. Security and PEN Testing
25. User Interface internationalization language Testing
26. Product Documentation: FAQs, User manual
    1. **Assumptions**

***General***

1. This effort estimation is based on the current understanding of the requirements and assumptions. The efforts might change based on the changes in scope or understanding of requirements in discovery phase
2. PropertyGuru to provide KT and access to existing Automation Framework and Testing tools
3. The test plan for existing features is present and would be shared with the partner

***Technical***

1. Happiest Minds will baseline the Guruland codebase from feature, security, and performance perspective before the start of the development and will use the baseline as reference for developing new microservice. Any new feature changes would impact on the project effort and schedule
2. PropertyGuru has followed DDD framework to arrive at the candidate domains for microservices in the RFP
3. PropertyGuru will provide security and performance benchmarks for the new services.
4. PropertyGuru shall be taking care of any changes to be made to the external systems (Salesforce, NetSuite, Workato) due to rollout of new microservices
5. As the existing tech stack is PHP Symfony and microservices will be developed using Springboot/Go, there would be no code reusability between the old monolithic and new microservice based applications
6. New microservices will be built based on the Business Requirement Document (BRD) prepared during the initial couple of sprints of the project. The existing PHP code will be read to augment the feature use case with specific validations and logic
7. Unit test scripts of the monolith application will be updated or added new case only for the integration changes to the new microservice end points
8. The CI/CD pipeline for the monolith application will be used for the integration changes introduced by the new microservice integration in the ecosystem
9. The user session management will continue as part of the monolith application. Only User Principal will be passed as JWT token to the new microservices
10. The request-response contract to the Presentation layer of the monolith will not be changed. The BFF layer in the new microservices ecosystem will conform to the existing contract
11. The existing error code and localized messages will be used. The localization will be done only of the new resources for the existing marketplaces
12. The performance or functional automated test results will be reported in separate spreadsheets by collating the data manually across test suites. A unified dashboard application will not be developed to share these reports
13. The solution and estimation are based on considering AWS cloud platform. The solution architecture and effort might change if a different cloud platform is chosen
14. PropertyGuru team will provide test data for executing migration scripts and performance testing
15. Tenants will be managed in the existing Identity Provider service. Also, the user association with a tenant
16. The existing IDP service has support for Oauth2.0 to protect the new microservice endpoints
17. We will use the existing html email templates for email creation. We will evaluate this further during discovery phase and recommend a suitable approach considering all available options such as AWS Pinpoint, Segment, Braze
18. Existing email servers will be used for all the email notifications
19. Smoke/Sanity test suites are automated by the PropertyGuru QA team, the QA team will evaluate further if any modifications need to be made during discovery phase

Discovery Phase

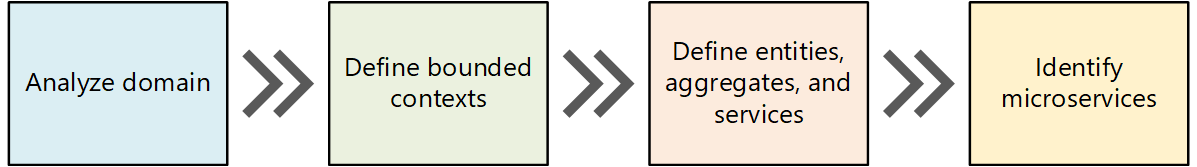
1. The discovery phase will be used to repurpose the proposal document with the changes in solution, effort, execution details and cost. The detailed documentation on the analysis of a curved-out domain and its DB schema definition will be done during the initial sprints of the implementation phase
2. In the discovery phase, PropertyGuru will provide walk through, CI/CD pipeline walk through, existing deployment landscape, technical sessions to validate technical challenges and all available functional and design documents
   1. **Dependencies on PropertyGuru**

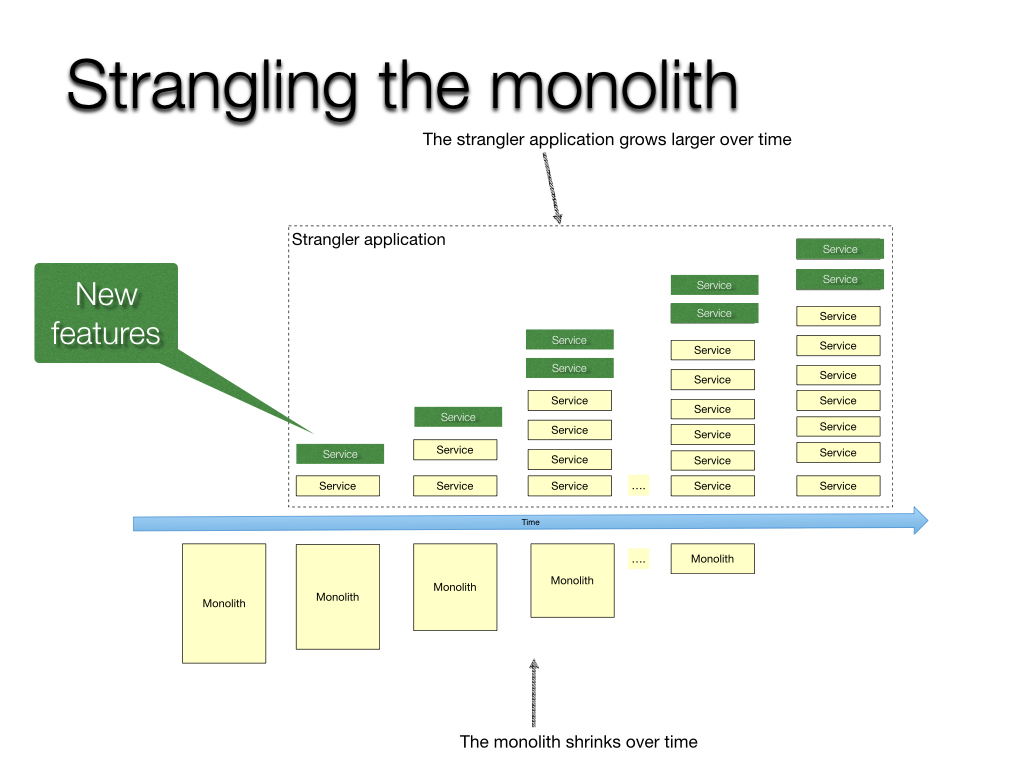
|  |  |
| --- | --- |
| **Dependency Factors** | **To be available by** |
| SMEs, Product owner of the application for requirement clarification, and review of the user stories | Start of the project |
| Technical SPOC for feedback and clarifications | Start of the project |
| Development Infrastructure Setup & Network Provisioning for development activities (on AWS or equivalent) | Start of the project |
| Knowledge transition of the existing architecture, codebase, schema, and features walkthrough | Discovery phase |
| Access to existing requirement documents, solution or design document, API documentation, DB schema, Test Strategy/Plan/Test cases, Automation scripts | Discovery phase |
| AWS account details | Start of the project |
| Licenses for commercial software | Start of the project |
| Email/Notification format and content | Start of the project |
| Code repository GitHub, Jira, SonarQube and Jenkins access. | Start of the project |
| Feedback for queries and clarification | Within 2 days |
| Feedback for milestone deliverables | Within 5 days |
| Review UAT Test Case document | As needed during the project |

1. **Solution Design And Approach**
   1. **Design considerations**

The proposed technical approaches for migration and to-be architecture after migration are based on the requirements specified in the RFP.

Following are the design considerations to modernize PropertyGuru monolith application to microservices architecture:

1. **Domain Driven Design (DDD)** – Decomposition of PropertyGuru monolith application backend into microservices will be based on business domain as required. List of business services listed in section 2.1 are the subdomains or core domains which are considered for carving out the as individual microservices.
   1. DDD framework as defined below will be followed to arrive the candidate domains for microservices.
   2. However, as we review the code and other artefacts we may add/modify services in consultation with PropertyGuru based on DDD principles and refine the overall microservices architecture. This may be due to a change in service boundaries, too many dependencies of components intertwined with each other etc.
2. **Business as usual** – Application migration will follow the [Strangler Fig](https://martinfowler.com/bliki/StranglerFigApplication.html) pattern to allow the migration of existing functionalities from monolith to microservices in an incremental manner, thus satisfying the requirement to continue the business as usual despite the changes to services. This approach will allow both monolith and microservices to exist together and provides a unified view of the application to the front-end layer until all services become independent. This enables us to avoid business disruptions and at the same time achieve migration in an incremental fashion as depicted in the below diagram.



*Diagram from:* [*https://microservices.io/patterns/refactoring/strangler-application.html*](https://microservices.io/patterns/refactoring/strangler-application.html)

1. **Multi-tenancy** – Each new microservice enables multi-tenancy by having tenant specific database schema connected through a single cluster of applications deployed as Software-As-A-Service model. The tenant specific configurations are pulled from configuration management system by deployment framework through Infrastructure-As-Code (IAC).
2. **Messaging and Integrations** – Establishing the APIs and deciding the best API communication pattern (asynchronous, synchronous, event driven) is key consideration for design and scalability of APIs. Events are generated to synchronize domain assets and their relationships `between monolith and new microservices. Event mechanism is leveraged to communicated between microservices to maintain the domain asset integrity.

* **Scalability** – Services will be scaled horizontally. The scaling factors will be based on inputs from the considerations such as observability metrics, performance metrics and business inputs. Current numbers with respect to the scale includes 3.5 million real estate listings, 40 million property seekers and 64000 active real estate agents.

1. **Security –** A layered security approach with controls at various levels such as network, API access level, user access, data security. Network segmentation, isolation of microservices to reduce blast radius, monitoring and logging, container security, Secure software development lifecycle addressing threat modeling of system are some of key design considerations.
2. **Resilience –** Automatic recovery from failure mechanisms (retries, circuit breakers, fallbacks), horizontal scaling, automating change managements and measuring against KPIs provided by PropertyGuru as inputs will be important considerations for the Resilience workload design.
3. **Data management considerations –** breaking, restructuring databases into single independent databases.
   1. **Modernization Strategy**

As per 6R’s model, the modernization of PropertyGuru from monolithic to microservices is based on Refactoring approach (Rewriting / Decoupling of applications) out the following approached in this model:

* 1. Rehosting (Lift and Shift)
  2. Replatforming (Lift and Reshape)
  3. Repurchasing (Replace – Drop & shop)
  4. Refactoring (Re-writing / Decoupling applications)
  5. Retire / Not moving
  6. Retire / Decommission

The modernization strategy is prepared based on Strangler pattern with the above design considerations. New strangled microservices will be built for each domain over a period. The domains are identified based on Domain Driven Design (DDD). While building the strangled microservices, the monolithic application co-exists with the new microservices. Once all the microservices are rolled out, the old system can be terminated. The following aspects will be followed in the design and implementation of the modernization activity.

1. Analyze and Identify
2. Transformation
   1. Blueprint design approach
   2. Key Technicalities
      1. Routing request to new microservice
      2. Multi-tenancy
      3. Data integrity and rollback
      4. Data sync from new microservice to old system
3. Retirement of old system
4. **Analyze & Identify**

In analysis phase, the domain specific individual services are identified based on the bounded context and those will be documented. The first service to be migrated will be decided based on the following factors:

* 1. Domain which has moderate level of difficulty to handle with the monolith application
  2. Reasonable level of risks calculated with mitigation

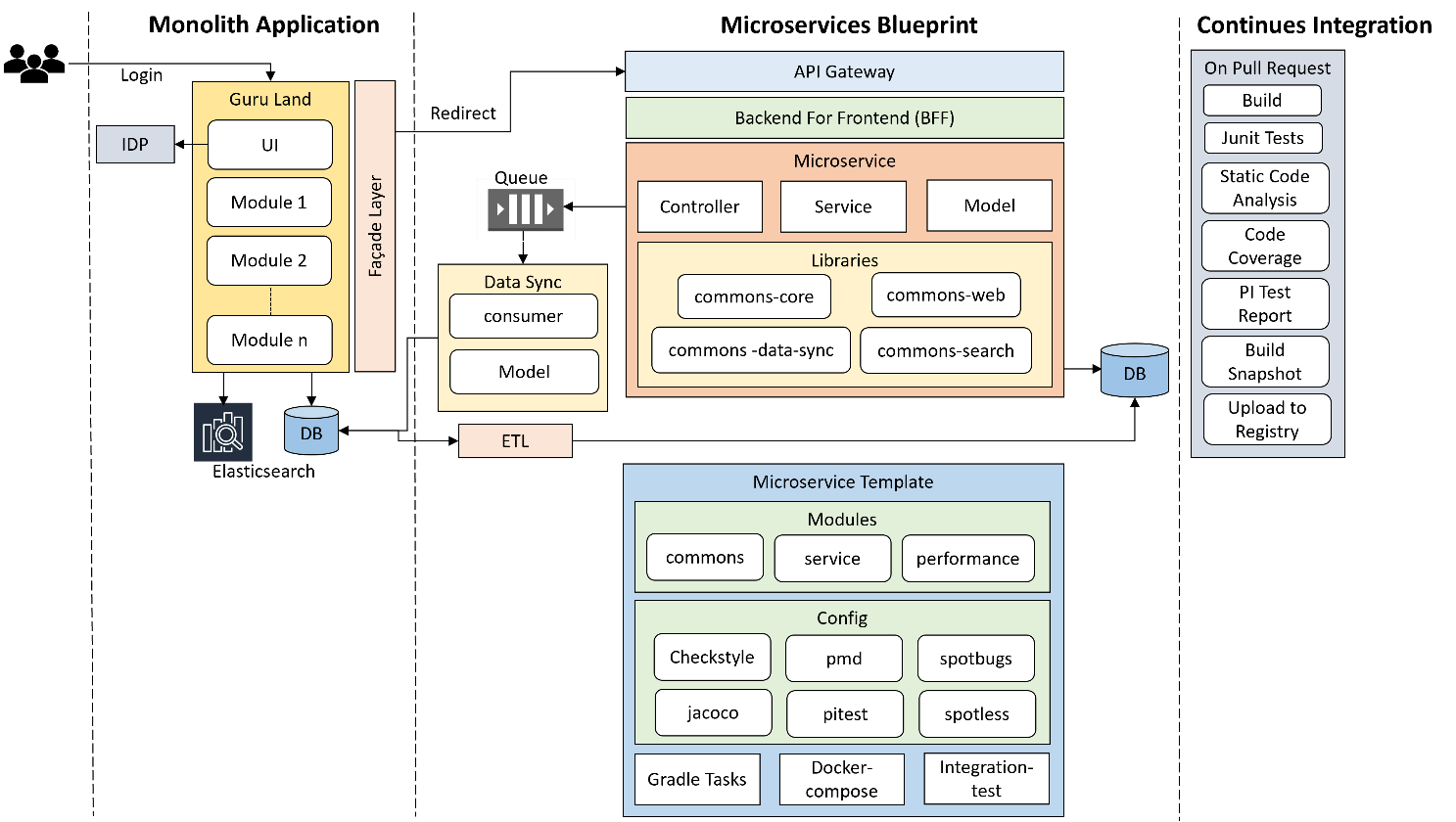
***Services with higher level of complexity***

1. The level of complexity of the domain will be identified on analysis
2. The complexity depends on the
   * 1. no. of relationships with other domains
     2. No. functional operations on the domain
     3. No. of transactions on the domain and transactions triggered on the related domains
     4. Whether the related domains are already as a new service or with the old system
     5. Volume of data to be synced between old system and new microservices
     6. Identified risks on various layers of the application
   1. The analysis and dependencies will be documented categorically on data structure, application design, integration points
3. **Transformation**

***Transformation considerations:***

* 1. Event driven architecture to generate events from the new microservice to propagate the data to old system to maintain the data integrity of other domain features
  2. ETL process to migrate old data of domain service specific schema. This ETL process to be created for each new domain specific microservices
  3. Transactions in the old system for the migrated domain generate events either from the service layer or as database triggers. Using these events, the new microservice data is maintained for integrity
  4. Continue the user login from old system and pass on the JWT token to the new microservice requests

Prepare the development ground based on the Transformation Blueprint as defined below:



***Blueprint design approach***

1. Define schema and model for a domain as documented during the analysis
2. Define API contracts using OpenAPI Specification
3. Create a microservice template such that the same template code is used to quickly bootstrap upcoming new microservices. This microservice template will be maintained as a separate repo in GIT
4. Liquibase scripts database manage schema changes and data seeding
5. Build common libraries incrementally by designing reusable components, utils. These common libraries will be added as dependency to the newly developed microservices
6. Build infrastructure to generate events for the domain from the microservice
7. Build a domain agnostic microservice to consume, process and update the data in old system for these generated events. Message processing design to process events from different domain services
8. Create a Backend For Frontend (BFF) layer to orchestrate the UI or Mobile client data contract across multiple microservices

***Microservice Template***

Microservice template will have the following things added incrementally during the development of the first microservice.

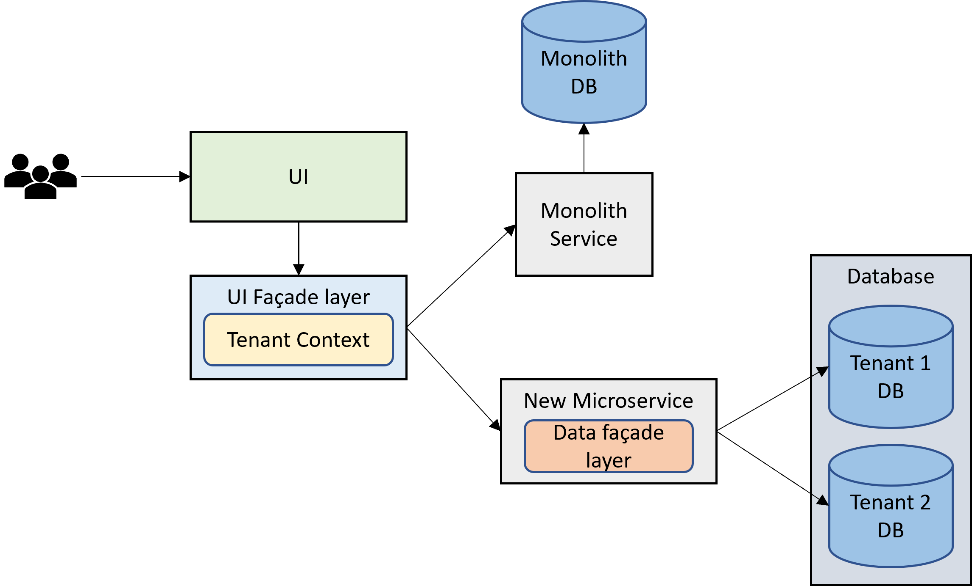
* 1. Gradle build files with settings, plugins, dependencies, and tasks to build, assemble, verify, and test
  2. Configuration for code quality tools such as Spotless, Checkstyle, PMD, Spotbugs, Jacoco, PI test. These tools will be executed on Verify Gradle task. The same will be configured in the configuration management tool on raise of a PR
  3. Docker-compose and Docker file having the required tools to help create the container for local testing
  4. Basic infrastructure to read configurations for external systems like Database, Cache, Message Queues etc.
  5. Standard modules such as common, service, and perf with Foo domain implementation of Controller, Service and Dao classes. Junit test cases for these Foo implementations

***CI on Pull Request***

1. Every Pull Request of the development requires a set of tasks to be performed as quality gates before preparing the deployment artifact
2. Development configurations and Gradle tasks will be prepared for quality gates execution
3. CI initiates the build, unit tests, integration tests, Static Code Analysis, Code Coverage, PI tests as part of the quality gate
4. The artifact generated post compliance of all the quality checks will be uploaded to Container Registry
5. CD scripts prepared will be used to deploy the artifacts on to the respective environment

***Key Technicalities***

1. **Routing Request to new microservice**
   * 1. The communication between the presentation layer to service layer can be of two types:
        1. Through a call to the REST API served by API Gateway
        2. Making direct call to database through data layer components built in the old system
     2. A new façade layer will be built between the UI layer and API gateway by extending the old system
     3. On successful login the tenant specific information will be passed on as part of user request to create tenant context
     4. In façade layer, create a tenant context with all the tenant configurations for each user session. It facilitates the routing to old system data layer calls or to API gateway using the tenant context
     5. Based on the tenant context route the downstream call to either to new service endpoint or to the existing flow or endpoint
     6. Routing at the API gateway will be done based on the API URI defined for old API and new API. The tenant information will be propagated to new microservices over request headers
2. **Multi-Tenancy**
3. Single application supporting multiple marketplaces as multiple tenants
4. Each microservice will have a separate database for each tenant
5. A data façade layer will be created as part of the microservices foundation to identify the tenant specific datastore to connect to the tenant database
6. The new façade layer created in the old system will identify and create the tenant content
7. The tenant context will be propagated to the microservices using which the tenant resources will be identified



***Note:*** Architecture supports having a separate schema for each marketplace/tenant within a single database. The single database approach provides optimized infrastructure usage. The proposal here is to have independent Database for each marketplace for each microservice.

1. **Data integrity and rollback.**
   * 1. The data of the migrated domain will be managed in the new microservice database. And the other domain data will be continued to manage in old system
     2. The integrity of data between the new microservice and old system will be maintained by syncing the data between old system and new microservice through event mechanism
     3. Each new micro service creation will have separate migration scripts to migrate data from old system for the specified domain
     4. Data from multiple marketplaces will be migrated to new microservice to the tenant specific schema. ~~A robust data migration process will be built to have hassle free migration~~
2. **Data Sync from new microservice to old system**
3. A new Data sync microservice will be created to manage the event generation and event processing infrastructure
4. Tenant specific events are generated and consumed over message queue
5. Separate message consumers are created for each microservice to update the data in the old system
6. New model hierarchy as per the old system object relations will be created to process and update the data in old system
7. A lot of business logic is involved in generating data integrated model before update
8. A robust message exchange infrastructure will be built to handle the failovers to maintain the data integrity at the old system
9. **Retirement of old system**
   1. On migration of the domains as microservices, the service layer of the old system can be retired
   2. The service layer and data layer component will be removed in packaging the old system
   3. As the connection to the old database from the service layer is cut off, the old database can be terminated
   4. Post retirement of the old system, the presentation layer component will be deployed using the Guruland repo
   5. The deployment of the presentation layer component will be integrated as part of the new CI/CD pipeline
   6. The old CI/CD pipeline also can be retired
   7. **Layered Architecture**

The proposed High level System Architecture (To-be architecture) of PropertyGuru system to implement the modernization strategy. The system will be organized in a layered approach for clear separation of responsibilities. The layered approach promotes re-usability, scalability, maintainability, and fault tolerance as layers/components have loosely coupled with each other. The advantage being, that the layers can be implemented, tested, and deployed independently.

Graphical user interface, application

Description automatically generated

***API Layer***

* PropertyGuru platform features are decomposed into multiple microservices which can developed, tested, and deployed independently
* Existing PropertyGuru API Gateway will be leveraged for all the new backend services which are mainly consumed by Web client applications
* All tailor-made composite APIs for the business requirements will be exposed through API Gateway
* Platform core will be built to manage the microservice foundation services such as Security, Data Access, Caching, Error handling and Logging
* Application Security
  + API will be secured through JWT token, which the PropertyGuru clients will pass over auth headers on each request
  + API will be authorized using auth tokens for each user based on their roles and permissions.
  + OWASP Top 10 security flaws and vulnerabilities will be addressed through static code analysis integrated as part of CI
  + Current PropertyGuru application Security based on OTP send through SMS or email will be continued
  + All communications are over SSL, which secures the request-response with data encryption

***Microservices***

All the PropertyGuru domain features will be implemented using micro-services architecture principles. This would mean all domain functions will be built as a self-contained service, which will be responsible for addressing the concerns for that specific domain functions.

***Data Layer***

* All structured transactional data of PropertyGuru will be stored in relational database and the unstructured data in the NoSQL database like the old system.
* SQL
  + 400 odd PropertyGuru tables will be categorized for each domain and retire unused ones as part of this modernization
  + Relational database is used for transactional data store
  + Independent database will be created for each microservice per tenant
  + Enables ease of querying requirements on strong relational model
  + Database instances with support for fail-over through master-slave mode
* MongoDB
  + Continue to use the current PropertyGuru User API data also include additional domains depending on the need during the modernization
  + Extensible schema, flexible schema, and horizontal scaling requirements
  + Service requiring purpose-built requirements such as Document storage
* File Store on cloud and Local file server
  + Cloud storage will be used to store all the archive data. Cloud storage has capabilities such as scalability, data availability, security, and performance
* Cache
  + Frequently accessed data will be stored in an in-memory storage, such that the data can be served faster, by cutting down the database server round trips. Especially, messages and at its attributes will be cached for faster retrieval
  + Redis Cache and Memcache shall be used as in memory database storing various data structures such as Strings, hashes, lists, sets, and key-value pairs
  1. **Technical Risks and Mitigation**

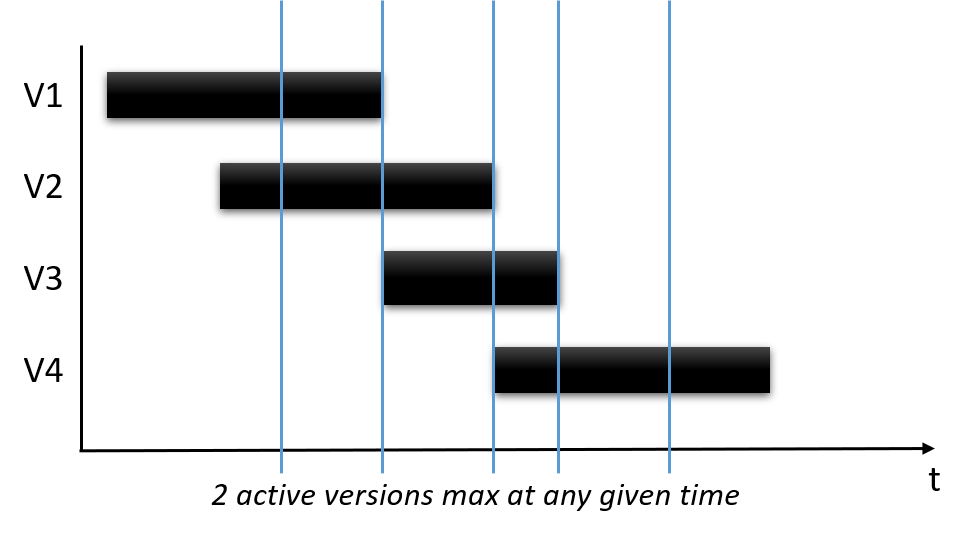
|  |  |
| --- | --- |
| **Risk Item** | **Mitigation** |
| When existing Identify provider service doesn’t support tenant management and associating user with tenants. | New module needs to be developed to enable tenant management and associating user with tenants. |
| Data migration of multiple marketplaces from marketplace specific database to new microservice will add more load on the source database. This may affect the operations of the existing system. | The migration will be performed at non-peak hours for each marketplace. A scheduled downtime also can be planned. |
| Trigger failure during generation at source or while processing at the target. Huge overhead on database. | Avoid using triggers as much as possible. As an alternate update Guruland implementation to generate events from service layer. |
| Data from new microservice will be consumed over message events in old system. To process the message, create new object relational models as existing in old system. A lot of business logic is involved in generating data integrated model before update. This new implementation may be error prone. | A rigorous testing of the existing system for regression needs to be performed on the old system. |
| Integration with external systems such as Netsuite, Salesforce and Workato requires some configuration changes to consume the new endpoints with the ability to identify whether to use the old or new endpoint. There may be feasibility issues on the external systems to make some customizations or configurations. | A new façade layer needs to be introduced at the PropertyGuru integration point. |
| Data integrity issues between old system and new microservices. | Periodic automated audit needs to be performed to check the correctness and integrity of data |

**Note**: above mentioned are an indicative list of risks that we pre-empt at this point of time however we shall be establishing a thorough risk identification & monitoring process and Early warning system which shall be leveraged for identifying, monitoring & managing all risks during the project execution.

**API Backward compatibility**

For APIs to be backward compatible we follow “TWO in Production” approach

Deploy and support two versions of an API endpoint and its operations that provide variations of the same functionality but do not have to be compatible with each other. Update and decommission the versions in a rolling, overlapping fashion.



Above approach supports the following

* Allow the services and the client to follow different life cycles so that a provider can roll out a new API version without breaking FrontEnd clients using the previous API version.
* Guarantee that API changes do not lead to undetected backwards-compatibility problems between clients and the services.
* Ensure the ability to roll back if a new API version is designed badly.

Data Syc

1. **Testing and Validation**

Following lifecycle shall be followed for Quality Assurance in the E2E QA Team

Graphical user interface

Description automatically generated with low confidence

* 1. **Baselining of Artifacts**

Team shall baseline all the references and the available documentation along with brainstorming with the customer stakeholders to ensure all available artefacts are baselined at the start of the project.

In case of AGILE projects where the architecture and designs evolve, necessary processes definitions shall be identified and concluded during this phase so that everyone in the team is clear on what is expected at each phase of the SDLC.

Happiest Minds’ Simple Design guidelines shall be followed in these cases.

* 1. **Test Requirement Analysis**
* Happiest Minds will document Test scenarios and perform Functional Testing, Integration Testing and End to End Testing that has been described in scope of work. Following is the approach which will be taken for functional testing of the system
* Smoke testing will be performed on the below PropertyGuru Applications to ensure the existing happy flow scenarios are functioning as-is:

1. Including iProperty (“iPP”) in Malaysia
2. Batdongsan (“BDS”) in Vietnam

This is primarily done to ensure that when we migrate the common User services, they do not impact the existing functionality

* 1. **User Scenario Analysis**
* We would be analysing various user scenarios based on how various roles who would be using the features as mentioned in Requirements sections for

Listing Management Service - For property managers, the ability to manage and update property listings quickly and easily is essential. Test include adding new properties, updating existing listings, and deleting properties that are no longer available. For real estate agents, the ability to view and manage their listings is important. Tests include filtering and searching for listings, updating listing details, and receiving notifications when a listing is updated or has an inquiry.

Listing Search – For potential buyers or renters, the ability to search for properties based on their specific criteria is crucial. Test include filtering by location, price, property type, and other relevant features.

Agent Subscription – For real estate agents, the ability to subscribe to the service and manage their subscription is important. Test include signing up for a subscription, managing subscription details, and receiving notifications when the subscription is about to expire.

Agent Management – For property managers, the ability to manage and communicate with agents is critical. Test include adding and deleting agents, assigning properties to agents, and sending notifications to agents.

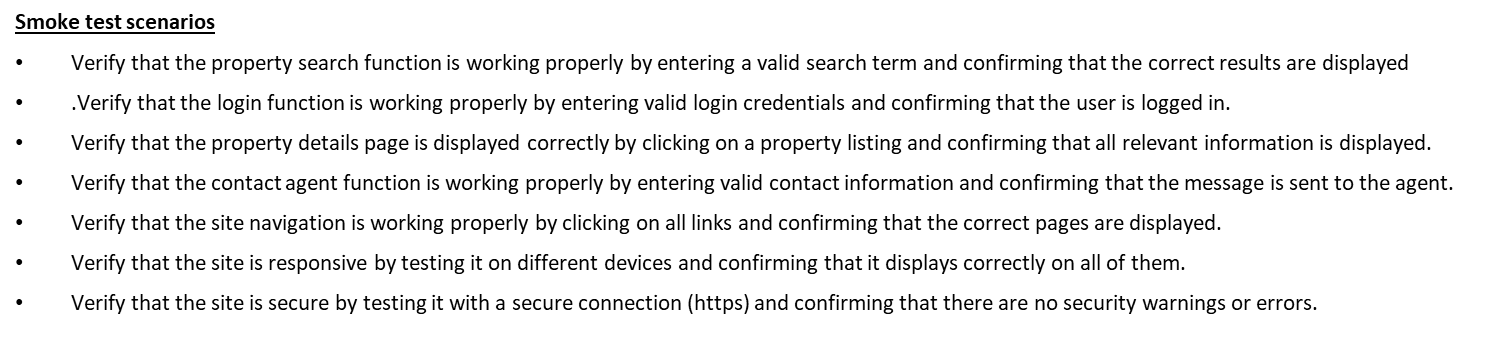
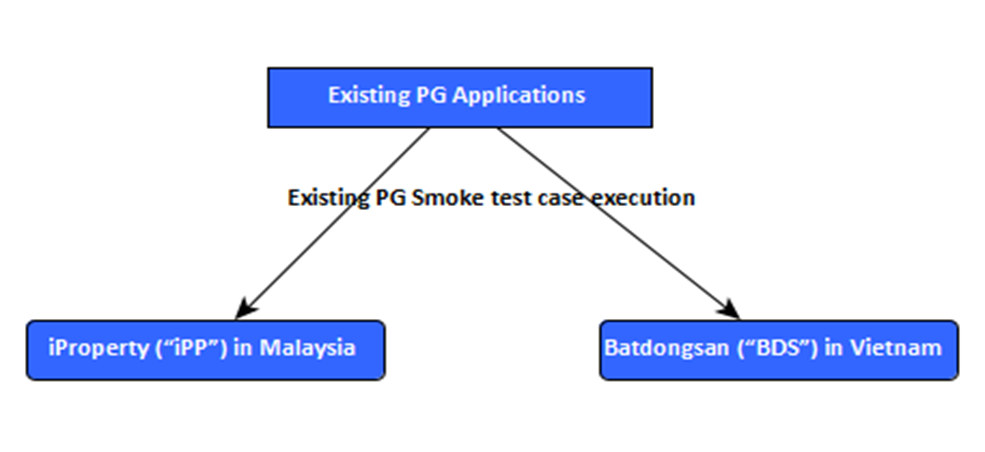
Users Service – For all users, the ability to manage their account details and preferences is important. Test include creating an account, updating account details and resetting passwords.

Mobile BFF – For users who prefer to use the mobile app, the ability to access all of the features of the web application is essential. Test include logging in, searching for properties, and managing listings.

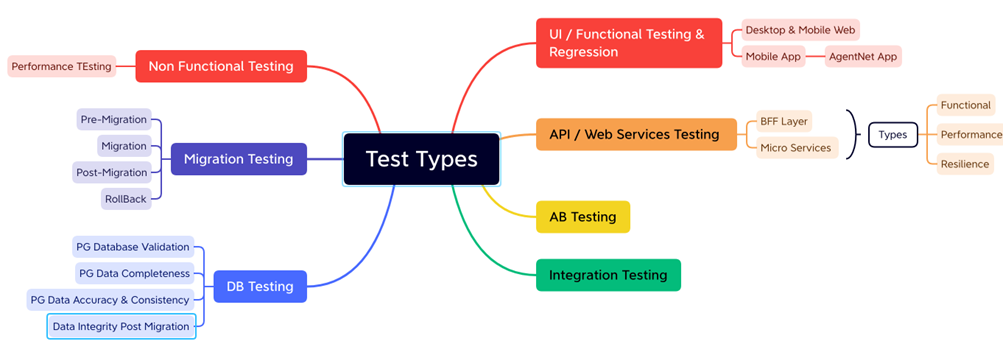
Admin Service (Frontend) User Engagement – Enquiry, Advertising Developer (Database Dependencies), Info Service Email Template Management and use the existing end-to-end test scenarios as part of good test coverage. Testing the application as per business logic and with focus on end user perspective would be considered.

* 1. **Test Cases Design**
* ***New features/requirements:*** we would be documenting detailed test cases to be used for execution using the Test Scenarios identified from the previous steps. Test cases would be prioritized as P1, P2 based on the criticality of the test case.
  + P1 test cases include all the Critical requirement of a particular feature/requirement
  + P2 test cases include major functional part as well as negative flows to validate behaviour and error/warning messages thrown by the application
* ***Regression features/requirements:*** Existing feature API Testcases would be provided by PropertyGuru Team and the new enhancement features API testing would be taken care by Happiest Minds Team. In case testcases are not available only high-level test cases would be documented to cover the existing features to ensure new feature implementation has not introduced any defects in the application.
  1. **Type of testing**

For existing PropertyGuru applications (iProperty (“iPP”) in Malaysia & Batdongsan (“BDS”) in Vietnam PropertyGuru) QA team will validate smoke test scenarios to ensure the existing feature are working as expected.



Below mentioned are the various testing types that would be considered during testcase design for new PropertyGuru requirement



|  |  |
| --- | --- |
| **Test Methodology** | **Description** |
| UI /Functional Testing | The functional testing shall be carried out as per the identified requirements for Listing Management Service, Listing Search ,Agent Subscription ,Agent Management ,Users Service ,User Engagement – Enquiry ,Advertising ,Info Service ,Email Management in the requirement documents / brainstorming with the engineering teams and product managers / standards that are planned to be adhered to. In areas where there are enhancements / feature additions / refactoring planned on the core components, QA will study the as-is behavior in any of the existing systems and ensure the as-is functionality is intact wherever the requirements are ported without changes from as-is (Existing) to to-be (new implementation) systems. For areas where there are changes (Modify & Delete) to the existing functionality, QA will study the impact of the change and identify test scenarios accordingly |
| AB Testing | Validation will be done using Sixpack testing framework provided by PropertyGuru, QA team will Install Sixpack, set up our experiment and its variations, including the metrics needs to be measured, Add the necessary code to PropertyGuru website or application to track conversions and send data to Sixpack and execute it. Analyze the results using Sixpack’s built-in reporting tools and which variation performed better and provide statistical significance testing.  QA will validate various PropertyGuru new and old application scenarios that can help improve the user experience and increase conversion rates.  Testing Different PropertyGuru Homepage Layouts: Testing different homepage layouts to determine which design drives more user engagement and conversions.  Testing Search Filters of PropertyGuru: validate different search filters to determine which ones lead to more successful searches and property inquiries.  Testing PropertyGuru Details Page Layout: Testing the layout and format of the property details page, testing whether a property video increases engagement compared to a photo slideshow.  Testing Lead Capture Forms: Testing the length and content of lead capture forms can help determine the optimal number of fields and types of questions to increase the likelihood of user conversion.  Testing User Experience: Testing different user experience features, such as chatbots or property comparison tools to improve overall user experience. |
| BFF Mobile Application Testing | Validate the integration of Mobile BFF & Agentnet Apps with new corresponding micro- services .QA will validate the functionality of mobile application on latest IOS and Android devices.  All API’s available for mobile application would be automated as part of API automation task |
| API Testing | Testing the existing features and testing the in scoped features.  API Testing would be performed on Swagger / Postman as suggested by PropertyGuru.  QA Team shall identify all the APIs (Internal & Third Party) and identify all the changes in terms if Request and Response Data Parameters, HTTP Header changes for encoding etc. and identify the test cases to be verified from existing set of API Test Cases from PropertyGuru  All Application Programming Interfaces exposed as a part of the system to the external or within the subsystem will be subjected to validation for response. REST API testing is performed at a message layer. API testing is used to determine whether APIs return the correct response (in the expected format) for a broad range of feasible requests and also react properly to edge cases such as failures and unexpected/extreme inputs. Test case design for API testing will be based on   * Return value – Return value based on input condition. * Does not return anything – When there is no return value, behaviour of API on the system to be checked. * Trigger some other API/event/interrupt – If output of an API triggers some event or interrupt, then those events and interrupt listeners should be tracked. * Update data structure – Updating data structure will have some outcome or effect on the system, and that should be authenticated. * Modify certain resources – If API call modifies some resources, then it should be validated by accessing respective resources.   API testing should be first type of testing that should be considered for Test Automation as most of the other sub system and the application working depends on the response of the exposed APIs |
| Integration Testing | Platform Integration points for each of the components will be identified and the test scenarios for the Integration testing will be designed. Main focus will be validating the component ability to integrate into the complete Platform and adherence to the data flow and the control flow paths of the Platform as designed for PropertyGuru. |
| DB Testing | PropertyGuru Database validation – QA will validate the integrity, accuracy, and consistency of data during the migration process.  PropertyGuru Data completeness: QA will ensure that all the necessary data is migrated to the new database, and no data is lost during the migration process.  PropertyGuru Data accuracy & Consistency: QA will ensure the data is accurate, and there are no errors or inconsistencies in the data and ensure that the data is consistent with the original source, and there are no discrepancies between the original data and the migrated data.  PropertyGuru migrate with New microservice: QA will ensure that each microservice is migrated with individual database created and validate the accuracy and consistency of data.  Data integrity: Ensure that the data is complete, accurate, and consistent, and there are no violations of data integrity rules, such as constraints, foreign keys, or unique indexes.  Performance & security validation: Ensure that the migrated database performs as expected, and there are no performance issues that may affect the overall performance of the application and ensure that the migrated database is secure and meets all the security requirements, such as access controls, authentication, and authorization.  Data validation: QA will ensure that the migrated data is validated against business rules, and there are no data quality issues that may impact the business processes. |
| Migration Testing | System testing will be performed in this case with all the data, which are used in PropertyGuru legacy application, and the new data as well. Existing functionality will be verified along with the new/modified functionality of PropertyGuru.  Migration testing will include testing with old PropertyGuru data, new data, or a combination of both, old features (unchanged features), and the new features, Different phases of the Migration test which will be performed for PropertyGuru applications.  **Pre-Migration Testing**  QA team will set testing activities which would be performed as a part of the Pre-Migration test after discussion with concerned stakeholders.  QA team will Perform data mapping between legacy and the new application – for each type of data in the legacy application compare its relevant type in the new application and then map them – Higher level mapping.  QA team will execute test cases, test scenarios, and use cases for new conditions in the new applications.  **Migration Testing**  QA team will validate PropertyGuru application meets the expectations of the end-users after the migration process.  The QA team will validate the user experience is not affected after the migration process.  QA team will validate all functionality and features are working as expected after the migration process to the new/existing users.  QA team will validate all data that has been migrated accurately from the old PropertyGuru database to the new one.  **Post Migration Testing**  QA team will validate PropertyGuru application end-to-end system testing is performed in the testing environment. All relevant identified test cases, test scenarios, use cases with PropertyGuru legacy data as well as a new set of data.  Backward Compatibility Verification  QA team will validate the new PropertyGuru system supports the functionality supported in earlier versions along with the new one.  **Rollback Testing**  Migration failure at any point of time during migration, system will be roll back to the legacy system and resume its function quickly without impacting the users and the functionality supported earlier.  QA team will validate applications that are able to rollback database changes made during the migration process to the previous state.  QA team will validate application is able to handle any issues related to data loss or corruption during the rollback process.  QA team will validate the rollback testing scenarios help to ensure that the application is able to rollback successfully in case of any issues during the migration process and recover to the previous state without any negative impact on the application’s functionality, performance, or user experience. |
| Performance Testing | Performance of the application/system/APIs decomposed out of PropertyGuru should be similar and improved when compared to the corresponding monolith implementation.  The performance testing will be conducted according to the performance benchmarks (existing performance data will be provided by the PropertyGuru team during the discovery phase) and will be used as the benchmark and come up with performance strategy accordingly. Performance team will ensure that the application does not degrade in performance compared to existing data.  Performance Testing shall focus on the following aspects.  **Load Testing**  The entire solution is tested against the expected normal/peak load conditions (by simulating expected concurrent user loads & workload patterns) to verify whether the application can meet desired performance objectives.  This is done on a single instance without any load balancing.  Objective is to identify the peak load; one instance of the system can take. This can be a vital input for capacity planning of the infrastructure.  **Payload Testing**  The system is tested against the data which is ingested with large payloads based on the identified boundaries. The payload is passed with maximum length for each of the parameters that the application accepts.  **Stress Testing**  The entire solution is tested with gradually increasing load until it reaches its breaking/saturation point in terms of response time, throughput & system resource utilization levels. This is done on a single instance without any load balancing.  Objective is to identify the peak load; one instance of the system can take. This can be a vital input for capacity planning of the infrastructure.  **Endurance Testing**  Subjecting the application to expected user load over a long period of time (8 Hrs., 12 Hrs., 24 Hrs. etc.). Focused on validating the performance characteristics of the application when subjected to normal production workload over an extended period of time and coming up with the system maintenance plan.  This is done on a single instance without any load balancing during Functional and Performance testing and once on full setup (including the final application) as part of E2E Testing.  **Scalability testing**  This is to determine the user limit for the application and ensure end user experience, under a high load depending on the application that is being tested, with different parameters: CPU usage, network usage or user experience.  **Reliability Testing**  This section covers the approach towards confirming the service continuity of the solution during local failures or during disasters. It also confirms the functionality of that portion of the PropertyGuru system which considers failsafe arrangement to ensure a) service continuity during system failure and b) load-balancing for optional utilization of infrastructure components. System availability is calculated by MTTF (mean time to failure) and MTBF (Mean time between failures).  Execution shall be carried out in Multiple Test Cycles incrementally. The Application Under Test (AUT) shall be loaded in steps (from 10% to 100%) and a full Automation Test Cycle shall be run to monitor the system behavior. The sequence shall be repeated with multiple loads.  **Backup Testing**  The system is tested to validate its ability to handle backup of different configurations, data, and state information on redundant systems to that they can be recovered when the system crashes or there is an ungraceful exit.  **Recovery or Failover Testing**  The system is tested to validate its ability to handle the load when one of the server/components fails.  It should also focus on ensuring there are no increase or decrease in the resource utilization (CPU, Memory etc.) when the system recovers.  There will be additional tests planned on the client and server side to measure the Throughput and Response times of all the content and pages for varied concurrency. |
| Security Testing | **Identification of Trust Boundaries**  Detailed Analysis shall be done and the trust boundaries at the sub system and the overall solution level. PropertyGuru security team will conduct security testing activities before System commissioning and we will ensure all issues/vulnerabilities reported in the application from the above security testing is addressed/fixed.  Test for common vulnerabilities such as weak passwords, insecure file permissions and upatched software.  Once the testing is completed, any issues or vulnerabilities identified during testing process should be addressed and fixed to ensure that the system is secure and does not pose a risk to the users. Coordinate with PropertyGuru security engineers as required to ensure that all security testing activities are performed correctly and any vulnerabilites are addressed in timely manner. |
| Microservice Validation | Functional Testing: Each microservice’s when developed, QA team would perform basic functional testing and cover all dependent scenarios for the specific microservice unit.  QA will cover all functionality by creating test cases that verify that the service performs as expected. This includes testing the API endpoints, input validation, data storage, and data retrieval.  Integration Testing: QA will validate the microservice’s integration with other microservices, APIs, and systems. This includes testing the microservice’s ability to communicate with other services and its ability to handle exceptions.  Performance Testing: QA will test microservice’s performance by simulating a realistic load on the service and measuring its response time, throughput, and scalability.  Security Testing: QA will test microservice’s security by identifying and fixing vulnerabilities, such as injection attacks, authentication and authorization issues, and data privacy violations.  Resilience Testing: QA will validate microservice’s resilience by simulating failures and testing its ability to recover from them. This includes testing the microservice’s ability to handle network outages, resource shortages, and other unexpected events.  QA will ensure that your microservice meets the necessary requirements and standards and delivers high-quality results. |

* 1. **API Automation**

Automated API tests provide much quicker test results and significantly accelerate development workflows; thus, it helps us speed up the feedback loop and catch issues faster. API automation will be implemented in a way in POSTMAN such that all the tests can be integrated with CI / CD Pipelines.

Approach:

**Identify the APIs:** The first step will be to identify the APIs that need to be automated. Identify the endpoint URL, request parameters, request headers, and expected response

**Postman Collection:** Create a Postman Collection and add all the APIs that need to be automated. Will group APIs into different folders based on their functionality.

**Create API Test Cases:** Create test cases for each API endpoint in the collection.

**Write Assertions:** Write assertions to validate the response data. Assertions will be added to the test cases using the Postman UI or by writing scripts.

**Execute the Test Suite:** Once the test cases and assertions are created, execute the entire test suite to ensure that all APIs are working as expected

**Integrate with CI/CD pipeline:** Integrate Postman with our CI/CD pipeline to run the test suite automatically every time the code is deployed.

**Generate Reports & Monitor APIs:** These reports will be used to track the test results and identify any issues that need to be addressed & use Postman’s monitoring feature to monitor the APIs for performance and uptime

* UAT Test case
* Sanity test cases
* Functional & Non-functional test cases that become the part of the regression test suite.
  1. **Test Types Vs Test Levels & Environments**

Test levels and environments will be determined for each test type of the Application / Subsystem. Each Sub System level test strategy will incorporate this information.

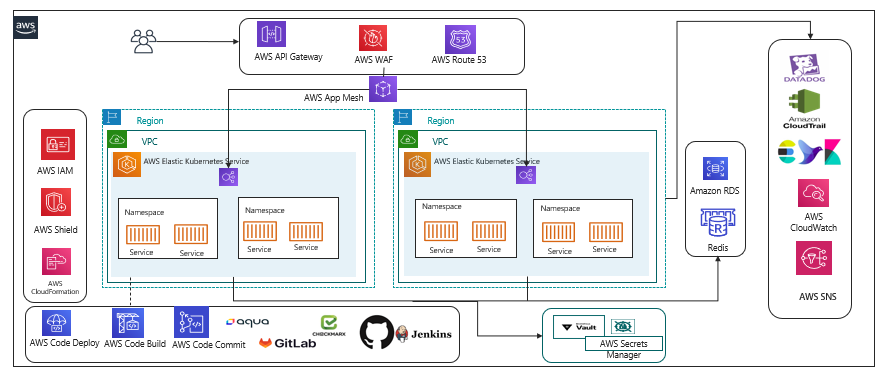
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Dev Environment** | **QA Environment** | **Bug Fixes** | **Before UAT** | **Pre- Production** | **Production** |
| Sprint Level | (Story Level) | (Story Level) |  |  |  |  |
| Rest API’s |  | Regression | Sanity | Regression |  |  |
| Web portal |  | Regression | Sanity | Regression | Sanity | Sanity |
| Mobile application |  | Regression | Sanity | Regression | Sanity | Sanity |
| Integration Testing |  | Regression | Sanity | Regression |  |  |
| Performance Testing |  |  |  | Yes |  |  |
| Security Testing |  |  |  | Yes |  |  |

* 1. **Test Execution & Defect Reporting**
* **New features/requirements testing:** Test cases would be executed on the dedicated test environment and devices. Defects would be categorized as Sev1, Sev2 & Sev3 and would be documented in the agreed upon defect tracking tool or excel document. (Sev 1 would be like application crash, functionality itself is not working. Sev2 would be functionality has issue but there is a work around. Sev3 would be cosmetic defects like UI issues, error messages are not informative, grammatical errors etc…).
* **Regression features/requirements testing:** Regression testing would be executed as per the test cases provided by PropertyGuru Team or documented by Happiest Minds.

1. **DevOps Strategy** 
   1. **Deployment Architecture for EKS**

The proposed deployment architecture is integrated with the Kubernetes platform deployed across the region. The Continuous Integration and Continuous Deployment being configured either with the proposed tools or with the existing tools. The services are isolated and deployed in the specific namespace in AWS account.

The below diagram and the tables gives the details about the tools integration.



|  |  |  |  |
| --- | --- | --- | --- |
| **Existing Tools** | **Recommended Tools** | **Description** | **Recommendation** |
| Elastic Kubernetes Service |  | EKS managed cluster is distributed across the two regions with High Availability  Autoscaling will be configured as part of the replica. |  |
| Elastic Container Registry |  | Base images are created for the containers to be deployed on EKS. We need the centralized storage for configure and uploading the non-prod and prod environments. |  |
| Terraform |  | Terraform for the IaC |  |
| AWS CloudFormation & Stackset Template | AWS CloudFormation StackSet | Templates configured for the provision of the AWS resources across multiple regions with a single cloud formation template. | AWS CloudFormation StackSet helps with a single cloud formation template deployed across the regions |
| Helm |  | Helm a package management tool will be configured for the release process and with the version control enabled.  Helm is used as release management tool integrated with the Kubernetes |  |
| AWS App Mesh /Istio | AWS App Mesh | App Mesh uses the open-source Envoy proxy, making it compatible with a wide range of AWS partners and open source tools.  App Mesh removes the need to update application code to change how monitoring data is collected or traffic is routed between services. App Mesh configures each service to export monitoring data and implements consistent communications control logic across your application. | AWS App Mesh configures each service to export monitoring data and implements consistent communications control logic across your application.  This makes it easy to quickly re-route network traffic when there are failures or when code changes need to be deployed. |
|  | Aqua | Aqua will be integrated as part of the platform to validate the prior image being uploaded on to the repository.  Security checks on the containers, images, code quality, vulnerability check will be integrated as part of the pipeline. | Aqua |
| Application Monitoring (EFK)  Cloud Watch | Datadog/CloudWatch | This is centralized and configured in the regions for extracting and maintaining the logs.  Additional Disk Volumes will be mounted as part of it so that the logs are always dumped to the volumes, extended without the changes affecting the system. | Datadog /CloudWatch |
| Infra Monitoring  CloudWatch | Datadog | Infra monitoring | Datadog: With the Instances monitoring, Application monitoring, Log Monitoring and all are available in a single dashboard. It’s also reducing to look on multiple dashboards for the monitoring purpose. |
| Datadog metrics monitoring |  | Datadog provides APM tools for monitoring the traces of the applications.  Datadog is tightly integrated with the Kubernetes pod for the monitoring of the internal execution along with the backend connection and the traces of the database execution.  Performance and the time taken for the execution of the queries are being traced and visualized at the centralized dashboard for monitoring. |  |
| Cloud Trail |  | Enables the auditing and monitoring API calls made with the AWS Accounts. |  |
| HashiCorp Vault Management | AWS Secret Manager | Secret management Integrates the secrets and the cert keys fetched from the Vault and integrated onto the services and pods during the deployment. | AWS Secret Manager: AWS secret manage is a managed services and pay as you go.  AWS Secret manager allows key rotation which is manual in HashiCorp.  HashiCorp is available as a open source and it does lay for the additional cost incurred in terms of scaling across  Overhead cost for maintaining the HashiCorp Vault. |
| **Security Protections**  AWS Shield  AWS GuardDuty  AWS WAF |  | AWS Shield for protection across the DDos attack.  Threat detection services which continuously monitors our AWS accounts  Web Application Firewall to protect from the common exploits threats. |  |
| AWS Elastic Cache for Redis |  | Create cross-Region read replica clusters for ElastiCache for Redis to enable low-latency reads.  Disaster recovery across AWS Regions. |  |
|  | GitLab |  | GitLab has the end-to-end feature integrated right from the code commit, code deployment, code review, branching strategy applied, Review and approve the code, promotion of the code, Pipeline oriented execution, Repository container inbuilt to store the images.  This makes it much more easy and integrated and helps for review and approve process as part of pipeline |
| AWS Managed build tools like codecommit, codebuild, codepipeline, codedeployment |  |  | AWS managed build tools for the Continuous Integration and Continuous Deployment in case GitLab in not opted for. |

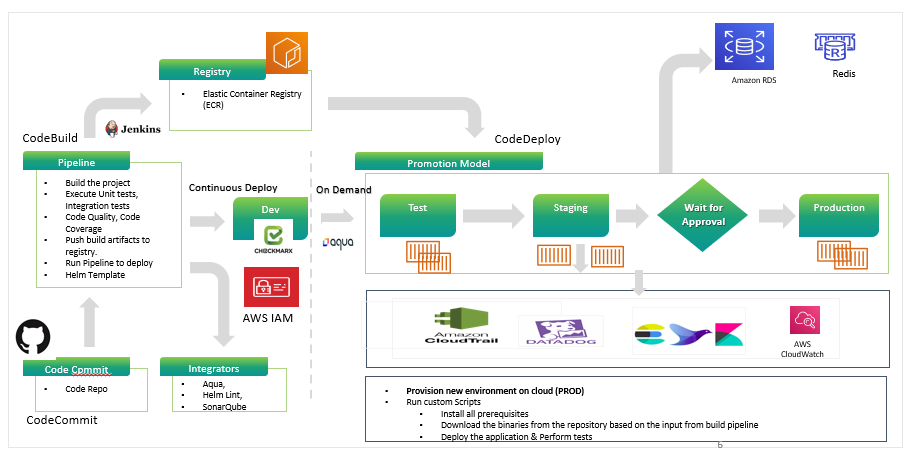
* 1. **AWS Governance**

AWS governance helps to support the organization workflow and to apply policies that are make the DevOps execution much more streamlined and aligned with the team for the faster execution.

The following are the few policies and roles will be utilized as part of AWS Cloud for the current execution.

|  |  |
| --- | --- |
| **Area** | **Description** |
| Authentication | Authentication mechanism will be enabled for the validation of the user identity validation across with the domain users configured. |
| Authorization | Access rights for the individuals are configured for the controlling of the access across the resources /services.  AWS Standard practices will be performed so that the project and the BU across the region are handled only by the respective projects. |
| Secret Management | The isolation of the access will be done through the Identity & Access management. |
| Identity Management and access control | Centrally managing users, security credentials such as access keys, and permissions that control which resources users can access. Using IAM  **Users** can access the assigned resources through the permission configured through the policies.  **Groups**:   * A group is being configured with the common group policies. * Policy applied are immediately traversed across all users in the group. * The groups are customized for each project specific.   **Roles**   * Configured for the complete isolation of the independent projects. * Roles are not configured with any users however they can be intake or consumed by the authorized users. * Different roles will be configured for the project specific execution.   **Policies:**  Policies are one of the important key factor to maintain the roles and the permission across the cloud.  AWS policies in specific the following will be configured for the platform:   * Access Control * Identity * Resource level permission * Service control policies * Access control List * Session Policies   Privilege Admin access will be configured so that the template layout, formation of the Cloud Formation Template of the services across the platform, validation of the users account and unique Id creation for the project, settings up the policies applied across the board. |
| Backup Plan | Regular backup process will be configured for the continuous delta data taken regularly for the Disaster Recovery. |
| Tagging | Enables the group of resources identified with the common names.  We can have the report generated and the custom commands executed as part of the report generation. |
| Governance | Policy includes to define risks, Alignment of the internal policies, Cloud formation template strategy customized, TLS version updated, capability embedded in all resources, Adherence to governance requirements. |
| Audit | Enable the independent or internal assessment for the cloud resources that are being part of the accounts. |
| Change Management | Approved changes being traversed across the production environment |
| Arch Council | To approve the specific tools for the evaluation and implementation |
| Cost Control | Asset Inventory management to optimize the cost. |
|  |  |

* 1. **DevOps CI/CD Pipeline**



The following are the steps execution as part of the pipeline the CICD pipeline is configured with all the requirement stages for the build execution:

* CICD Pipeline is configured with multi-stage build process for the pipeline build execution.
* Continuous Integration of the build process starts and triggers the staged build process.
* The triggered build process on compilation is successful proceed with the next gated check with the code quality.
* During the process the image is validated for all the security integration with the scanning using the tool.
* The image of having the vulnerability criteria falls into the criteria like high, medium and low grouping.
* The rules being set with the expectation for the code to pass in and to upload on to the Registry.
* On successful compilation the build is uploaded onto the Container Registry.
* The artifact is then called using the Continuous Integration process for the artifact deployment onto the K8 cluster.

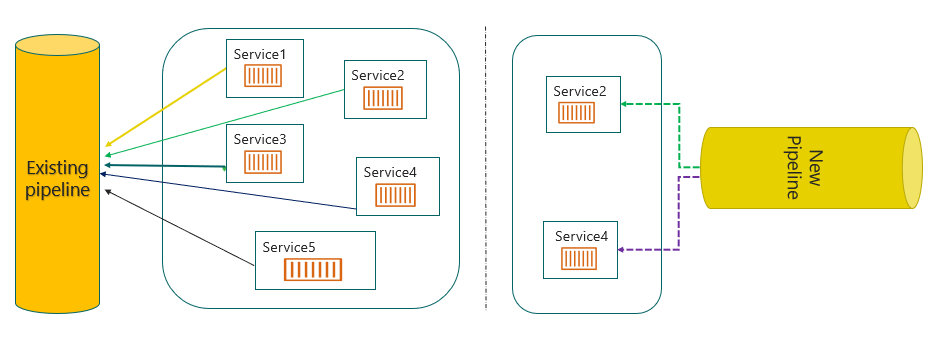
**Release process**

* The releases are done using the templates configured with the chart for the specific repos and the same can be reused across the board. The releases are version controlled and the roll back can be done at any point of time.
* All the artifacts are stored on the Container Repo and tagged with the defined labels.
* The promotion model is being set so that the same artifact can be promoted onto the next environment using the pipeline.
* Non prod environment setup done for doing their testing without disturbing any existing environment.
* Performance test scripts are integrated to run on the code deployed and validated.
* Alerts are being configured at all the levels in case of failure to trigger so that they’re completely monitored with the tools.

**Application and Infrastructure Monitoring**

* The application infrastructure monitoring is being done using the Datadog & CloudWatch integrated as part of the platform.
* The Datadog tool will be tightly integrated & the agent configured for the entire monitoring.
* The agent installed on the monitoring will be retrieving all the data and the logs related to the application and infrastructure.
* The APM (Application Performance Monitoring) will be integrated with the platform for the monitoring and all the services, and the pods will be integrated for the traces capture of the platform logs.
  1. **Strangler DevOps Pipeline**

Proposed Strangler DevOps pipeline for the existing as well as the new pipeline.



The strangler proposed DevOps pipeline works as follows:

* The monolithic and the new micro services configured will be running in parallel in the pipeline.
* The new micro service will be built independently from the existing Monolithic service.
* The new micro service as well as the existing monolithic service pipeline will have the services running in both the platform.
* The monolithic service that is split and built in new micro service is being tested
* The new micro services developed will be validated from the new cluster.
* The new micro services are validated for the parallel execution, working as expected then the service will be removed from the old pipeline.
* The cycle continues further for the other microservice split onto the new pipeline.
* The existing pipeline needed changes will be coordinated with the team so that there is no impact.
  1. **Branching strategy**

We are proposing the two strategies for the branching to be followed and the most common strategies followed:

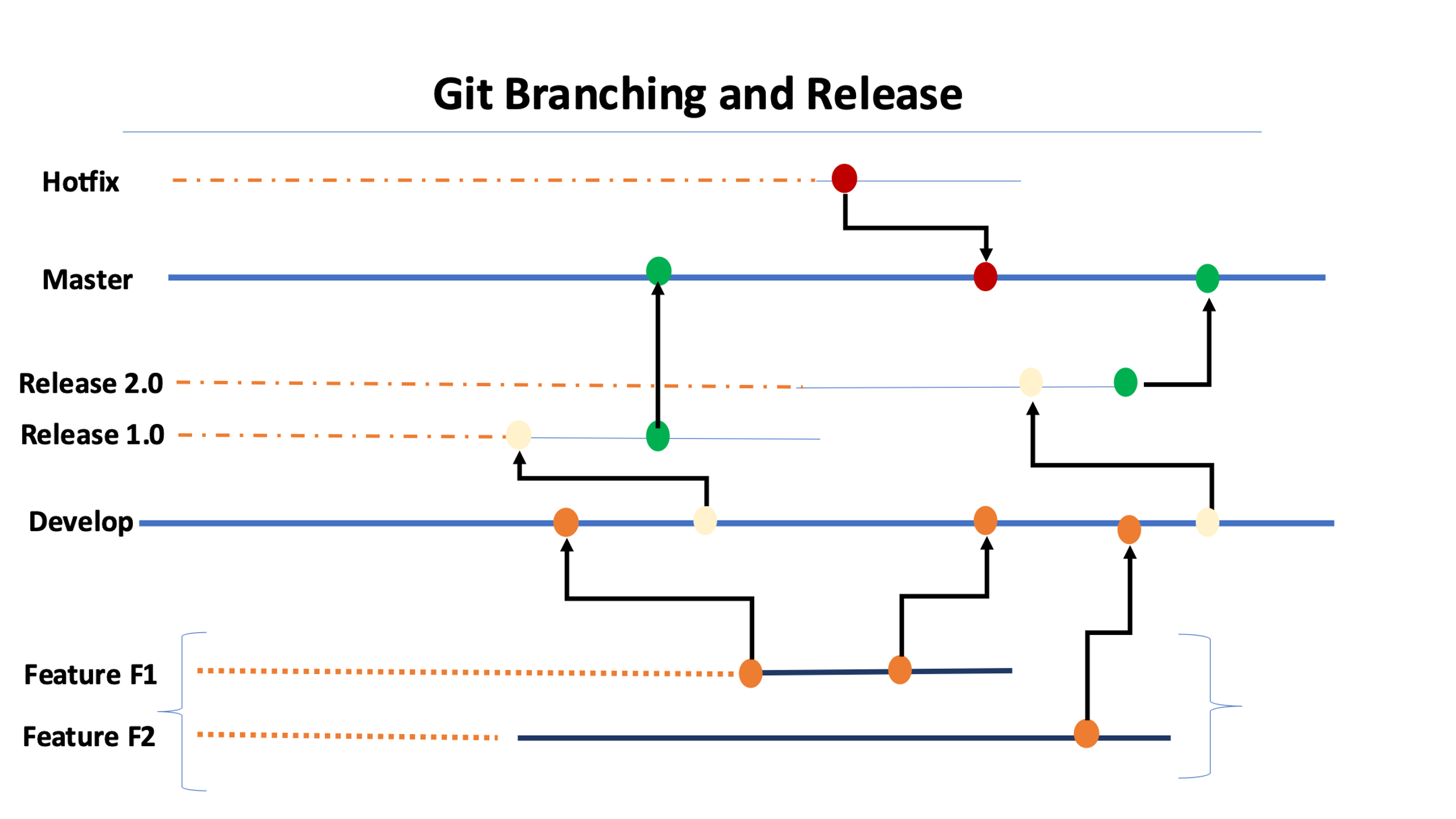
* GitLab Flow
* Trunk Based Development

**Option 1:**

**GitLab Flow:**

The branching strategy consist of the following branches:

* Main Branches
  + Master
  + Develop
* Support Branches
  + Release Candidate branch
  + Hotfix branch



**Proposed Flow:**

* Standard conventions are followed for the branching strategy so that the product is uniquely identified.
* The feature branches are short lived branches which are being developed and merged into the development branches.
* The development branches are then merged on to the supporting branches called as the Release branches.
* Tags are being created from the branches to do the releases.
* The release are then tagged based on the short outcomes of the releases.
* Finally approved changes are being merged back onto the Master branch and the deployment happens from this branch.
* The master branch is then again branched out for the new changes in development/feature to worked upon by developers.

The hotfix branches are created in case of any critical issues to be fixed on the production release being done from the master branch

**Option 2:**

**Trunk Based Development**

Chart, line chart

Description automatically generated

Recently the changes are with the strategy aligned to get the development with the short-lived branches so that we have faster releases. Trunk based development is a version control management practice where developers check-in their code to trunk or master branch. The developers finish the new work on the feature, and they merge the changes on to the new code trunk. There are challenges seen during this time with the conflict arising, Conflicts are increasingly complex as development teams grow and the code base scales. The trunk-based development model reduces these conflicts.

**Advantages**

* It allows continuous code integrati–n - In the trunk-based development model, there is a repository with a steady stream of commits flowing into the master branch.
* It ensures continuous code revi–w - Small commits of trunk-based development make code review a more efficient process. With small branches, developers can quickly see and review small changes.
* It enables consecutive production code releas–s - Developers should make frequent, daily merges to the master branch.
* It’s ready to deploy to production at any time.
* This gives the team agility to frequently deploy to production. It also ensures that the further goals of production releases to be made much more frequent than the current process.
* Quicker to deploy and resolve the challenges rather than waiting for the long cycle completion.
* More collaboration during the execution and work as a one team with fully owned responsibilities.

**Recommendati–n - Option 2** is recommended since most of the organizations are aligned with this strategy process due to frequent merges to the master branch.

This can be reconsidered during the discovery process based on further engagement.

* 1. **Release Management**

**Release Process**

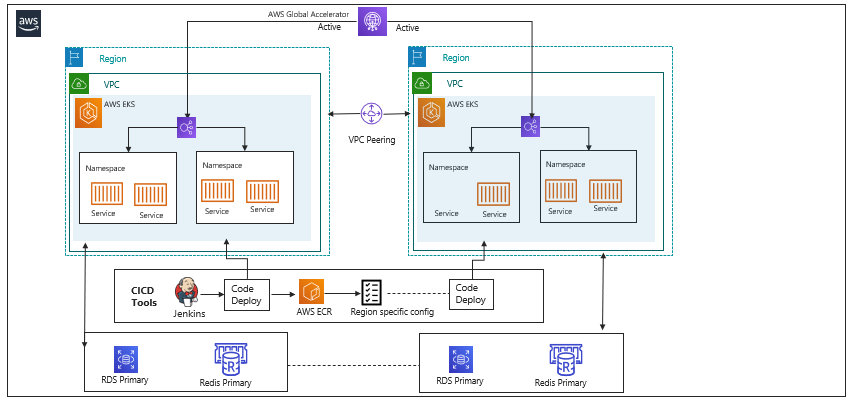
* The release process will be created with the managers & cab approval obtained prior the releases.
* Release processes are defined with the CI/CD Integrated
* Deployment will be done through the build tool.
* Frequent branches and the tagging process will be created for the deployment.
* Release will be tagged and stored as a Tag in the Repo.
* Hotfix branches will be created during the process of any critical fixes that need to be part of the deployment.
* Communicate to the Stake holders on the release complete
* The Helm chart will be used for the version control of the Releases.
* The Release will be deployed using the pipeline with the specific branching Tag created for the deployment.
* The Releases will be version controlled and also deployed from the specific pipeline.
* The promotion of artifacts will be done using pipeline.

**DevOps Cloud Security for Application**

* Security Virtual Private cloud will be configured.
* Security Rules for the private and public cloud configured.
* Access Identity and Management (IAM) Access Analyzer
* Key Management Service Integrated for encryption.
* S3 configuration and monitoring
* Security assessments on the firewall rules & configuration
* Access to the corresponding repos
  1. **Disaster Recovery**

**Please refer to the section** [**Deployment Architecture for EKS**](#_Deployment_Architecture_for) for the deployment.

Disaster Recovery diagram projected here is for two regions with active-active configuration.



The above diagram gives the solution running across the regions with the active mode of configuration done. The proposed model will ensure that all the required services are up and running.

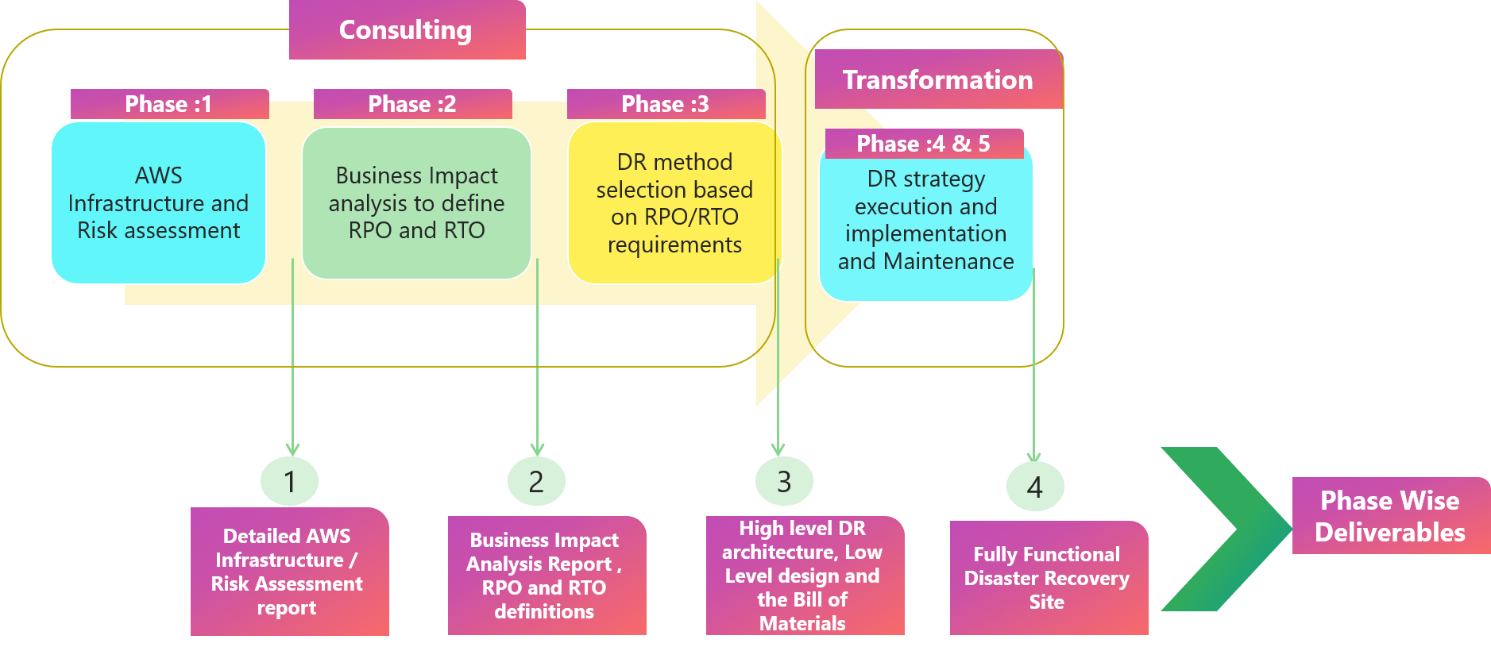
The backup and the pilot mode of execution will be integrated and executed during the process of execution.

The following are the flow will be enabled for the execution:

* Active- Active mode enabled for both the regions so that the recovery will be available always on with minimal or zero downtime.
* Real time execution is made available.
* Reduce the recovery time to the nearest zero down time.
* Backup frequency needs to be more due to the dependency on the recovery of current execution.
* The estimated backup will be depending on the RTO/RPO

The only disadvantage is that the cost of the AWS services is too high.

* 1. **Proposed Approach for Disaster Recovery**



In this architecture, we will build the realize and the dependencies applications in AWS:

* AWS will be connected through AWS Direct Connect/VPN for secure access.
* Application data will be cloned onto the target regions and continuous replication will be enabled.
* In this scenario, there are two parallel environments running for the platform.
* End user traffic will be routed using AWS managed services.
* Further, the responses sent from both the regions would be logged for analysis and validation.
* The platform will be tested in the AWS environment and will be cutover during a change window as per the scheduled.
* The DR process is being configured with the active – active mode where the clusters are being configured with the High availability.

**Continuous Integration and Continuous Deployment**

* Configuration code is configured in muti-region.
* Deployment of the infrastructure changes and configuration/code changes to be deployed at the same time with Primary Region.
* Analysis on the definition of RTO & RPO.
* Implement the DR method based on the previous results obtained.
* Code stored in the repo will be trigged in the pipeline for the active application deployment .

**Node Cluster Backup**

* EKS cluster nodes are configured in the multi region for the DR process.
* The node cluster will be configured in two regions for the distribution.
* Distributed pods deployment in both the regions across the cluster
* Actively monitor the active regions that are up and running.
* A template will be created for the deployment execution during the DR.

**Databases & Datastores**

* RDS distribution across regions will be configured during the initial setup.
* Database backups are configured regularly based on the discussion.
* The backup will be created using the AWS managed resources.
* DevOps will facilitate a Backup process for the regions.

**Other Options**

**Acti–e - Passive setup** where the passive will be available in the warm setup so that we can switch over the Egress during the fail-over.

* Here only one region will be active at any point in time.
* The process remains the same as that of the above with minimal changes.
* There is a minimal change made for the execution to make it Live for DR.
* Cutover will be performed with the analysis of the regions failover.

1. **Technology Stack – Indicative**

|  |  |
| --- | --- |
| **Feature** | **Technology** |
| Programming language and Framework | Typescript, JavaScript, React, Java 11 |
| Cloud | AWS, CloudFront, EKS, ECS, Redis, IAM, RDS, MYSQL |
| Authentication | Existing User Authentication module, HashiCorp Vault |
| Networking | AWS Global Accelerator (for Disaster Recovery), AWS WAF |
| API Gateway | AWS API Gateway |
| Microservices​ | NestJS for I/O bound functions and Java Springboot for CPU bound operations  Docker, Kubernetes |
| Database/Storage​ | MySQL, MongoDB, Elasticsearch |
| In memory Cache | Redis, Memcache, ElastiCache |
| CI/CD pipeline | Jenkins/GitHub/GitLab/AWS CodeCommit Pipeline, Docker, EKS, ECS, Kibana, EFK, Datadog, Prometheus, Terraform |
| QA Tools | Rest Assured (API Automation), Test Execution: SauceLab (Only for Mobile), |
| Performance Testing | JMeter |
| Web Server | NGinX |
| Messaging | RabbitMQ, SQS |
| Feature Toggle | Six Pack |

1. **Warranty and Maintenance Support** 
   1. **Scope and Requirements**
   2. **Support requirement**

* Setup of the warranty and maintenance application support team for the microservices developed by Happiest Minds.
* Establish the required Service Management process.
* Provide first point of contact for any P0 issues (system outage).
* Provide Lev–l - 2, 3 ongoing technical support.
  1. **Out of Scope**
* Support of any feature or services not developed by Happiest Minds
* Infrastructure, cloud, network, database, and security support
* Helpdesk, Level 1 Application Support
* Hardware support
* Field support
* Application enhancements and feature development
  1. **Definition of Priority**

The encountered issues will be classified as follows.

Blocker – P0

Critical – P1

Major – P2

Minor – P3

* 1. **Solution consideration**

|  |  |
| --- | --- |
| **Support team of PropertyGuru can reach to Happiest Minds support team** | * Jira for ticket tracking and management * Email for Updates * Dedicated mobile number for P0 issues. * Microsoft Teams or similar collaboration tool for regular connect, issue triage or address P0 incidents |
| **Support Language** | * English |
| **Support Window/Service Coverage** | * 24x7 coverage for production outages (P0) with on call support * 8x5 for P1, P2 and P3 tickets |
| **Monthly Tickets Volume** | This will be discussed during discovery phase how to prioritize the ticket and volume based on the capacity of the support team allocated |
| **Duration** | Starts from the deployment of 1st service and covers till 12 months from the date of completion of the contracted work |
| **SLA requirement** | The expected response time by support team would be:  P0 – 15 mins  P1 - 1BD (business day)  P2 - 3BD  P3 - 7BD  The expected resolution time by support and engineering team to fix the issue would be:  P0 - 1 BD  P1 - 8BD (business day)  P2 - 18BD  P3 - 28BD |

* 1. **Post Implementation – Support Operating Model**

Graphical user interface

Description automatically generated

* 1. **On Going Support Task details**

**Application Support**

* 1. **Level 2 Support** 
     1. Working with PropertyGuru support team to resolve the escalated issues.
     2. Troubleshooting and RCAs
     3. Change management coordination.
     4. SOP creation
     5. Patch Releases
  2. **Level 3 Support** 
     1. Complex troubleshooting & RCAs
     2. Change management
     3. SOP validation
     4. Availability, Change, Problem and Capacity Management
     5. Reporting and Recommendations
     6. Continuous Improvement Planning
  3. **Minor Enhancement** 
     1. Bug fixes related to features and services developed by Happiest Minds
  4. **Timeline**

Table, timeline

Description automatically generated

* 1. **Major Incident Management**

Diagram

Description automatically generated

* 1. **Assumptions**

1. Any feature enhancements /platform upgrade/new initiative/major release activities will have to be taken up as a Change management process or separate project.
2. There will be a single point of contact from PropertyGuru for this engagement, who will be working with Happiest Minds team for the execution of the engagement.
3. SLA clock will be paused during any responses/assistance expected from 3rd party providers’ / infrastructure support team(s) / Product team. Happiest Minds will not be responsible for any SLA impacted due to any delays caused by PropertyGuru ’s fulfiller teams; 3rd party providers/vendors for support resolution.
4. The team will be based out of Happiest Minds premises in one of our office locations in India
5. Ticket management system (Jira) and collaboration tools (other than MS Teams) will be provided by PropertyGuru
6. Access to production system, application or system logs, monitoring tools, issue reproduction steps will be provided by PropertyGuru .
7. Coordination with end user, any 3rd party vendor, infrastructure team, product team will be responsibility of PropertyGuru support SPOC
8. **Roles and responsibility. Below table details roles and responsibility amongst different stakeholders.**

|  |  |  |  |
| --- | --- | --- | --- |
| Scope | PropertyGuru | Happiest Minds Support Team | PropertyGuru Support team |
| L1 issues / access issues | **-** | **-** | **IC** |
| L2, L3 Application Incident | **-** | **AR** | **IC** |
| Incident Resolution for Application Platform |  | **AR** | **IC** |
| Security issues |  | **IC** | **AR** |
| AWS provisioning Support / DevOps | **-** | **I** | **AR** |
| Change management & Coordination | **AR** | **I** | **R** |
| Reporting | **IC** | **RA** | **R** |

* 1. **Indicative SLAs, Response and Resolution Times**

**Incident Management**

Happiest Minds application support team are only responsible for the issues/incident which can be resolved at L2/L3 level only.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Serial No | Request Type | Description | Response time | Resolution time | Communication method |
| 1 | Critical Incident – Severity 1 | System down, Impacts business | 15 mins | 1 BD (Business Day) | Ticket and **telephone** |
| 2 | Medium Incident – Severity 2 | System not performing as desired, impacts business | 1BD | 8BD | Ticket and **telephone** |
| 3 | Low Incident – Severity 3 | System not performing as desired, no business impact | 3BD | 18BD | Ticket |
| 4 | Non-priority Incident – Severity 4 | Minor bug, non-critical RFC | 7BD | 28BD | Ticket |

1. **Schedule and Milestones**

Following are the considerations for arriving at the proposed schedule, milestones & order of execution of these services.

* Initial milestones consider migration of less complex services and services which are more independent
* Services which are more complex and have more dependencies span across milestones and for these cases there would be intermediate deliverables, however service shall not be completely deployable
* Some of the cross-cutting concerns / common components like Authentication, Authorization shall be part of the initial milestones and be integrated with every service getting migrated

**Note:** below mentioned milestone details are more of an indicative list and this shall be revisited during the discovery phase.

Following is the list of high-level schedule and tentative milestones. This is an indicative one and more detailed and accurate project plan shall be prepared during the discovery phase (please refer to the activities & deliverables of the Discovery phase in the Section-D document)

**Project Start: T**

|  |  |  |
| --- | --- | --- |
| **Milestone** | **Tentative feature list** | **Schedule** |
| Milestone1 | * Listing search service * World API * Email template management service | T + 4 months |
| Milestone-2 | * Developer (Database Dependency) * Listing Management Service * User Engagement – Enquiry | T + 7 months |
| Milestone-3 | * Agent Subscription * User Service * Listing Management Service | T + 10 months |
| Milestone-4 | * User Service * Agent Management * Mobile BFF | T + 13 months |
| Milestone-5 | * Info Service * Agent Net | T + 15 months |
| Milestone-6 | * Advertising * Admin Service (Frontend) * Agent Net | T + 18 months |
| Milestone-7 | * Admin Service (Frontend) * My Activities | T + 20 months |
| Milestone-8 | * Stabilization Milestone * Regression testing of the new platform. * Defect fix & verification | T + 21 months |

**Milestone deliverables**

The following are the high-level deliverables for the above-mentioned milestones.

* Source code and binaries
* SonarQube reports on the delivered services.
* Test cases, test reports, test coverage reports.
* Design document for the services
* Deployment scripts and config documentation

At the end of each service migration (during intermediate stages), we shall ensure we have the new service working cohesively with the overall monolith system to cater to the respective use cases / business cases for the identified microservice.

**Note:**

* Each service migration shall involve the following activities.
  + Façade layer for abstraction of functionalities
  + Bi-directional data sync between the new service and the respective monolith module (for any rollback needs)
  + Persistence layer migration (DB separation)
  + Data migration
  + Any changes (if any) to Monolith to support the new service.
* Each milestone shall be followed by a UAT phase of 2 weeks where co-existence of monolith & new microservice shall be tested.
* Acceptance criteria for UAT is as mentioned in the below section. Any deviations shall be discussed with PropertyGuru and mutually discussed and agreed on the action items and any course corrections.
* After successful completion of UAT, the new service shall be deployed in the production environment followed by support.

**User Acceptance Testing Criteria for above deliverables**

* Unit test coverage of 100%
* Zero Security & Vulnerability defects in the SonarQube reports
* Acceptable levels for Schedule and Cost variance (+/- 10%)
* Passing 100% of the user acceptance test cases with Zero Critical (P0) and High (P1) severity bugs identified
* Defect Severity - Definition: Definition of Defect Severity and Priority are as below.

|  |  |
| --- | --- |
| **Defect Severity** | **Definition** |
| * P0 – Critical | * Defect may be a showstopper - that is, it stops the user from using the system further. |
| * P1 – High | * Defect occurs repeatedly and prevents the user from proceeding in the normal way, but a workaround exists. |
| * P2 – Medium | * A defect is isolated or does not stop the user from proceeding but is annoying and causing inconvenience. |
| * P3 – Low | * A defect that in no way affects the performance or functionality. E.g.: Aesthetic issues and grammatical errors in messages. |