ASK NICELY

WORK SAMPLE

https://asknicely.needlecompass.com

Developed by **BHUSHAN MAHESHWARI**

Work Samples

- Faced with an existing system that uses a coupled templating engine like Smarty or Twig, how would you go about planning and executing a strategy for migrating the product to a headless frontend? Assume an existing PHP backend REST API is already available or can be extended to support it.

- If we wanted to reduce our back-end reliance on PHP, replacing it with Typescript, how should we do it? (Alternatively, what would be most important to avoid?)

- How should we organize the communication between our front-end and back-end? Currently most of it is REST-ish, with a small percent documented in Swagger. One part uses GraphQL instead. We still have a number of PHP-rendered pages which we are gradually migrating to a Vue.js single-page app.

- You take over a legacy system that uses a Vue.js frontend but requires a push to its next evolution of wider test coverage and convention adoption for a team of Engineers. Assume test tooling such as Cypress or Cucumber are already available, but not deeply familiar to some of the team. How would you approach this evolution and socialize any changes to multiple teams?

- How can we double the productivity of our engineers (without trading job satisfaction, system availability, quality, or security)?

Understanding Work Samples

Strategize

Faced with an **existing** system that uses a coupled templating engine like **Smarty or Twig**, how would you go about planning and executing a strategy for **migrating** the product to a **headless frontend**? Assume an existing **PHP backend REST API** is already available or can be extended to support it.

Best Practices

If we wanted to **reduce** our back-end reliance on **PHP**, **replacing** it with **Typescript**, how should we do it? (Alternatively, what would be most **important to avoid**?)

Communicate

How should we organize the **communication** between our **front-end** and **back-end**? Currently most of it is **REST-ish**, with a small percent documented in **Swagger**. One part uses **GraphQL** instead. We still have a number of PHP-rendered pages which we are gradually **migrating** to a **Vue.js** single-page app.

Adoption

You take over a **legacy** system that uses a **Vue.js** frontend but requires a push to its next evolution of **wider test coverage** and **convention** adoption for a team of Engineers. Assume test tooling such as **Cypress** or **Cucumber** are already available, but not deeply familiar to some of the team. How would you approach this **evolution** and **socialize** any changes to multiple teams?

Work Smarter

How can we **double** the **productivity** of our engineers (without trading job satisfaction, system availability, quality, or security)?

Strategize

Faced with an existing system that uses a coupled templating engine like Smarty or Twig, how would you go about planning and executing a strategy for migrating the product to a headless frontend? Assume an existing PHP backend REST API is already available or can be extended to support it.

Questions

- Core need for this migration?
- Do I need to create a working prototype along with devising the strategy? Additionally, I believe we are just talking the front end here?
- What would be product's complexity, criticality, and # of real users?
- When you say strategy, does it include roadmaps and timelines?
- What are the integrations, touch points, or possible dependencies?
- Do I need to consider the team distribution and skills of developers, or I can assume that is already taken care of?
- Is there a preferred front-end technology/framework? Backend PHP - Laravel?
- If I can ask about existing features/functionalities/modules? Anything to be added/removed as part of migration?
- How do you want me to handle authentication/authorization?

Work Smarter

How can we double the productivity of our engineers (without trading job satisfaction, system availability, quality, or security)?

Questions

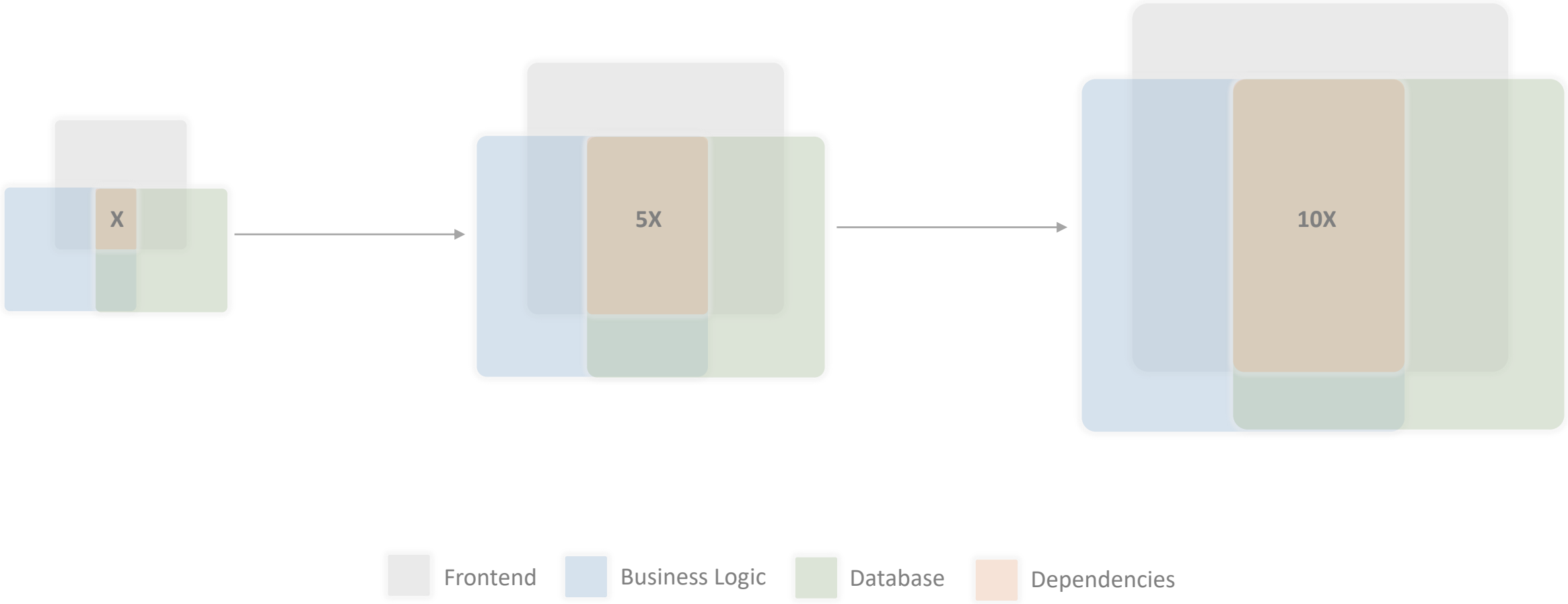
- I can assume that this needs to be done at an org level?
- Any concerns or reports on current productivity?
- Any analysis that has been done to understand how engineers are utilizing their time and energy?
- Are we using any tools to monitor, like Teams Admin, Viva insights?
- Are we considering this for Hybrid, Remote or Office?
- On an average, what is the current routine for any engineer?
- Are all the engineers at one location, or its geo-spread (what ratio)?
- What is the expected output of this sample work – a simple list of to-dos/recommendations from renowned source, my own experience or a strategy comprising a holistic view in a PowerPoint presentation

Strategize

Faced with an **existing** system that uses a coupled templating engine like **Smarty or Twig**, how would you go about planning and executing a strategy for **migrating** the product to a **headless frontend**? Assume an existing **PHP backend REST API** is already available or can be extended to support it.

Understanding Problem

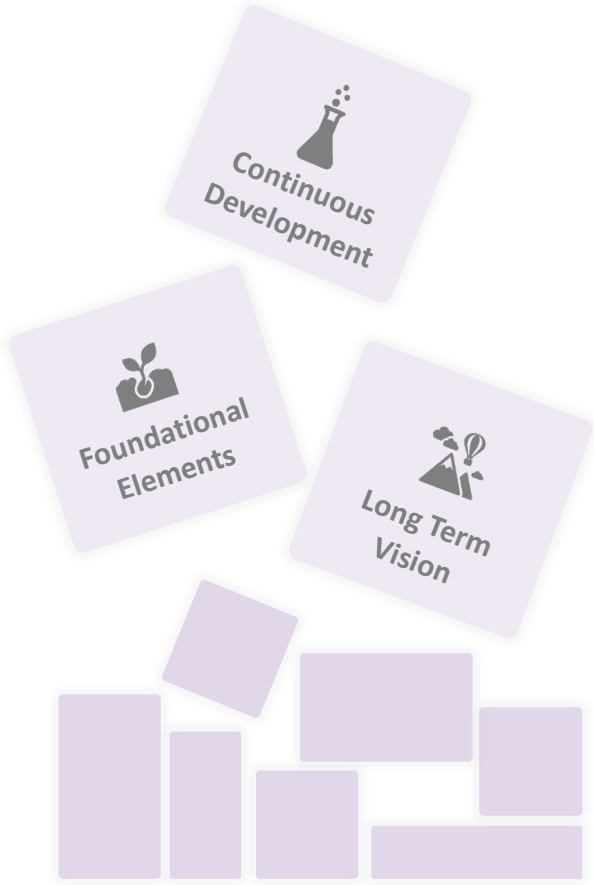
and need for the digital transformation



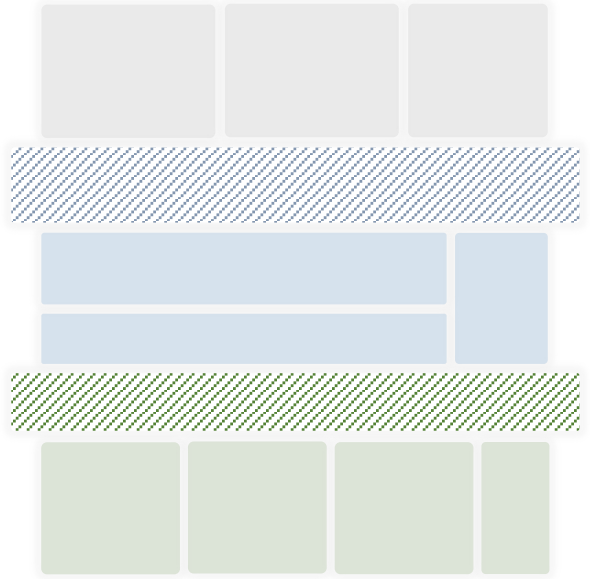
Migration Journey

from decoupled to headless

Current State



Core Building Blocks



Future State

Building Blocks

and the underlying strategy

Leadership | Technical Team | Product Team

Long-term
Planning
roadmaps, user-research, etc.

Roadmaps and Timelines
aligned timelines with org goals, milestones, progressive adoption, documentation, etc.

Customer Centric
empathy, target for omnichannel, change management, accessibility, etc.

Continuous
Development
start simple, scalability, risk analysis, etc.

Roadblocks Mitigation
downtime, hosting, infrastructure, external dependencies, parallel releases, etc.

Understanding what Matters
clean up features, add most asked features, performance, mobile-first, SSR, etc.

Code Mapping
.tpl files, {include} to components, block to slots, Vue-CDN progressively, {php} tags, etc.

Best of the Breeds
adoption to automation, cloud, testing frameworks, PWA, DevOps, etc.

Foundational
Elements
best practices, uniformity, security, etc.

Design System
UI-toolkits (leverage/extend), theming, predefined CSS, icons, layouts, etc.

Core Libraries
modern JS (Vue), common-utils, components, auth-libs, project shell, caching, etc.

Work Smarter

How can we **double** the **productivity** of our engineers (without trading job satisfaction, system availability, quality, or security)?

Missing Pieces of Productivity

and how to complete the puzzle

Work

+

Agile Culture

Silos vs collaboration, chaos vs innovation, hero-es vs team

Capability Building

accountability, empathy, focused, be authentic, collective ownership

Conversations

formal and informal, relationship, ignite queries, skill sharing, listening

Growth Mindset

retrospective + result-oriented, seek help, feedback, one day at a time

Building Trust

learn from failures, empowerment, transparency, work ethics, respect

=

Work **Smarter**

Appendix - Architecture https://asknicely.needlecompass.com

ANALYTICS (TELEMETRY)

GOOGLE: Firebase

AZURE: Application Insights

AZURE CLOUD HOSTING

CDN HOSTING

APP SERVICES

REPOSITORIES AND DEVOPS

GITHUB ACTIONS: GitHub

TESTING FRAMEWORKS

UNIT/COMPONENT: cypress.io

END TO END: Playwright

HEADLESS FRONTEND

DESIGN SYSTEM: BULMA.io

MODERN JS

API LAYER ENDPOINTS

REST APIS: node JS

DATA SOURCES AND DATA API

DOCUMENT DB: mongoDB

Thank you!