


U.S. Digital Services Playbook checklist and data collected

Play Checklist and Key Questions	Response / Artifacts
<div> PLAY 1 Understand what people need Pavel Khozhainov</div> <div>Checklist<ol style="list-style-type: none">1. Early in the project, spend time with current and prospective users of the service2. Use a range of qualitative and quantitative research methods to determine people's goals, needs, and behaviors; be thoughtful about the time spent3. Test prototypes of solutions with real people, in the field if possible4. Document the findings about user goals, needs, behaviors, and preferences5. Share findings with the team and agency leadership6. Create a prioritized list of tasks the user is trying to accomplish, also known as "user stories"7. As the digital service is being built, regularly test it with potential users to ensure it meets people's needs</div> <div>Key Questions<ol style="list-style-type: none">1. Who are your primary users?2. What user needs will this service address?3. Why does the user want or need this service?4. Which people will have the most difficulty with the service?5. Which research methods were used?6. What were the key findings?7. How were the findings documented? Where can future team members access the documentation?8. How often are you testing with real people?</div>	<div>Checklist<ol style="list-style-type: none">1. 01. Interviews2. 03. User Personas 04. User Stories & Scenarios3. 04. Local user interviews4. 02. User Needs5. UX Design Process6. 04. User Stories & Scenarios7. 01. Interviews</div> <div>Key Questions<ol style="list-style-type: none">1. Foster parents, Biological parents, Relatives (special case of Foster parents), Authorized representative for Biological parents2. Secured communication with caseworkers. Ability to trace and extract information. Ability to identify the closest facility with better services provided.3. Today there is no single way of secured communication. No archive. No export function. Existing public database of the Foster care facilities is not very useful for Foster parents and Biological parents.4. Biological parents (literacy challenges, Spanish-speaking only)5. User interviews, Personas, Scenarios for usability testing6. User expectation for privacy, ability to trace and export data from secured communication7. All user meetings are stored in both mp4 and text form and all are in open access for the team.8. We test Axure prototype versions with local users on a daily basis. We test it with SME on a weekly basis.</div>



PLAY 2 Address the whole experience, from start to finish Pavel Khozhainov D ariia Iarmuratii

Checklist

1. Understand the different points at which people will interact with the service – both online and in person
2. Identify pain points in the current way users interact with the service, and prioritize these according to user needs
3. Design the digital parts of the service so that they are integrated with the offline touch points people use to interact with the service
4. Develop metrics that will measure how well the service is meeting user needs at each step of the service

Key Questions

1. What are the different ways (both online and offline) that people currently accomplish the task the digital service is designed to help with?
2. Where are user pain points in the current way people accomplish the task?
3. Where does this specific project fit into the larger way people currently obtain the service being offered?
4. What metrics will best indicate how well the service is working for its users?

Checklist

1. For anticipated online interaction, see 04. User Stories & Scenarios. For in person, we anticipate that users discover the service through their caseworker or from another parent (the website link would be shared). A caseworker can use the facility search while interacting with a parent regarding placement for their child.
2. Please see the discussion of User Needs, Pains, and Gains as described in 04. User Stories & Scenarios
3. UX Design Process
4. 06. Usability testing

Key Questions

1. Users communicate by phone with the case worker. Some users may know about the online facility database where they can look up the history of an organization (foster care agencies, adoption agencies, group homes). Some may use the public email to send messages. There is no current support for secured communication between biological parents and caseworkers.
2. Pain points highlight the need for a secure way to communicate with caseworkers. There is no way to find a facility based on distance from a preferred address. There is no single place to store communication with caseworkers in a secure manner.
3. For the biological parent, it is a new service, available 24 hours a day, 7 days a week, to use to communicate securely with the caseworker. It can support storing communication that currently is only in personal email or hard-copy documents. For the State caseworker, this new communication channel can be a part of a CWS modernization project - parents portal module. This module can communicate with the caseworker's own message system. The secured message center and map visualization components can be reused in other modules.
4. We have goal based metrics - for each basic scenario (use case) we receive a clear Boolean answer indicating whether the user was successful in goal achievement. Depending on deployment location for the full prototype, Google Analytics can also be installed to monitor usage patterns.



PLAY 3 Make it simple and intuitive Pavel Khozhainov Dariia Iarmuratii

Checklist

1. Use a simple and flexible design style guide for the service. Use the U.S. Web Design Standards as a default
2. Use the design style guide consistently for related digital services
3. Give users clear information about where they are in each step of the process
4. Follow accessibility best practices to ensure all people can use the service
5. Provide users with a way to exit and return later to complete the process
6. Use language that is familiar to the user and easy to understand
7. Use language and design consistently throughout the service, including online and offline touch points

Key Questions

1. What primary tasks are the user trying to accomplish?
2. Is the language as plain and universal as possible?
3. What languages is your service offered in?
4. If a user needs help while using the service, how do they go about getting it?
5. How does the service's design visually relate to other government services?

Checklist

1. <https://playbook.cio.gov/designstandards>
2. 08. Pages in Design
3. 08. Pages in Design
4. 08. Pages in Design
5. 08. Pages in Design
6. 09. Decisions made
7. Unclear message texts need to be verified and fixed

Key Questions

1. The primary tasks are: 1. Send and receive messages from caseworker. 2. Find an old message when needed using different parameters like keywords in message content or message recipient. 3. Find some facilities near the user's location (current or planned).
2. Given the target audience, the solution uses plain language (simple grammar) and currently supports American English (other languages can be supported).
3. Our service is currently offered in English. Other languages are in the product backlog.
4. The product backlog includes support for a generic help icon. Product usability testing included testing for the alignment of the task to the design. We also have an orientation video posted to a YouTube channel on the main page that helps the user to understand how the CWS Parent Portal works.
5. The US Web Design Standards have been followed.

Checklist



PLAY 4 Build the service using agile and iterative practices Leonid Marushevskiy

Checklist

1. Ship a functioning “minimum viable product” (MVP) that solves a core user need as soon as possible, no longer than three months from the beginning of the project, using a “beta” or “test” period if needed
2. Run usability tests frequently to see how well the service works and identify improvements that should be made
3. Ensure the individuals building the service communicate closely using techniques such as launch meetings, war rooms, daily standups, and team chat tools
4. Keep delivery teams small and focused; limit organizational layers that separate these teams from the business owners
5. Release features and improvements multiple times each month
6. Create a prioritized list of features and bugs, also known as the “feature backlog” and “bug backlog”
7. Use a source code version control system
8. Give the entire project team access to the issue tracker and version control system
9. Use code reviews to ensure quality

Key Questions

1. How long did it take to ship the MVP? If it hasn't shipped yet, when will it?
2. How long does it take for a production deployment?
3. How many days or weeks are in each iteration/sprint?
4. Which version control system is being used?
5. How are bugs tracked and tickets issued? What tool is used?
6. How is the feature backlog managed? What tool is used?
7. How often do you review and reprioritize the feature and bug backlog?
8. How do you collect user feedback during development? How is that feedback used to improve the service?
9. At each stage of usability testing, which gaps were identified in addressing user needs?

1. MVP of application prototype version 0.1 <<[Link ontagin GitHub](#)>> was shipped after Sprint 1 on May 29th. So it took 7 days.
2. We used interactive Axure wireframes for regular usability testing with users in the early stage of development. Based on feedback obtained through usability testing, we quickly incorporated changes to the wireframe over the course of a two-day development cycle for the Design Group.

When MVP became available after Sprint 1, we switched usability testing to the actual application prototype.

3. EngagePoint realized the importance of effective communication within the Scrum team. We used daily standups to share progress reports and to resolve any blockers and stoppers. We used [Skype](#) for instant communication between team members as well as group chats for different activities like Development, Deployment, and QA. In order to conduct Design Sessions with the ability to share screens and draw on the whiteboard, we have used [WebEx](#). Our Scrum team worked in one open space, so free form communication between team members occurred continuously. Atlassian JIRA was used for issue related communications. Documentation for the project was developed in Atlassian Confluence tool.
4. Team structure is described in [Additional Requirements \(17 items\)](#).
5. We used one-week Sprints over three weeks and delivered three versions of the application prototype. We realized that a short development cycle sped-up development and provided more control over development for the Product Manager.
6. We used Atlassian JIRA to manage our Agile board and Product backlog.
7. We used local GitLab repository as a main version control system for source code, then moved source code to a public GitHub repository.
8. The entire team had access to both JIRA and GitLab environments.
9. We believe that automated tools must be used heavily for code review. This minimizes efforts otherwise required for manual review. Automated code review was managed using SonarCube, which controlled the quality of the source code, caught simple coding bugs, validated code duplication, code coverage, etc.

In addition to automated code review, we practiced cross-team review. Two of most experienced team members reviewed all commits to the repository and provided comments on implementation.

Key Questions

See [Agile Approach Overview](#).

1. Seven days
2. *Not Applicable*
3. Five-day sprints
4. Git
5. Atlassian JIRA
6. Atlassian JIRA for Agile board
7. Once a week during sprint planning
8. We used Axure wireframe for initial prototyping, usability testing, collecting initial feedback.
9. During usability testing, we have identified following gaps:
 - a. the user wants to see the number of facilities which are available for current search criteria
 - b. the user wants to see distance from his location to facilities on the map and sort list of facilities by distance
 - c. the user wants to be aware that some filters for facilities are selected
 - d. the user wants to be able to quickly ask about a facility
 - e. the user wants to see contacts in a private inbox and be able to quickly compose a message to this contact
 - f. the user wants to be able to export messages in PDF format



PLAY 5 Structure budgets and contracts to support delivery Margreta Silverstone

Checklist

1. Budget includes research, discovery, and prototyping activities
2. Contract is structured to request frequent deliverables, not multi-month milestones
3. Contract is structured to hold vendors accountable to deliverables
4. Contract gives the government delivery team enough flexibility to adjust feature prioritization and delivery schedule as the project evolves
5. Contract ensures open source solutions are evaluated when technology choices are made
6. Contract specifies that software and data generated by third parties remains under our control, and can be reused and released to the public as appropriate and in accordance with the law
7. Contract allows us to use tools, services, and hosting from vendors with a variety of pricing models, including fixed fees and variable models like “pay-for-what-you-use” services
8. Contract specifies a warranty period where defects uncovered by the public are addressed by the vendor at no additional cost to the government
9. Contract includes a transition of services period and transition-out plan

Key Questions

1. What is the scope of the project? What are the key deliverables?
2. What are the milestones? How frequent are they?
3. What are the performance metrics defined in the contract (e.g., response time, system uptime, time period to address priority issues)?

Checklist

1. Internal budget includes allocation of staff time to do research, proof of concept and prototyping. EngagePoint uses JIRA to support the development effort, including the use of story points to assess the level of scope complexity. Overall monitoring of activity through JIRA and regular Agile approach meetings support the overall project tracking.
2. Not Applicable (NA)
3. NA
4. NA
5. NA
6. NA
7. NA
8. NA
9. NA

Key Questions

1. See the [RFI Scope](#)
2. [Agile](#) provides regular check points on activities
3. Performance metrics are currently tied to the [RFI success factors](#)



PLAY 6 Assign one leader and hold that person accountable Margreta Silverstone

Checklist

1. A product owner has been identified
2. All stakeholders agree that the product owner has the authority to assign tasks and make decisions about features and technical implementation details
3. The product owner has a product management background with technical experience to assess alternatives and weigh tradeoffs
4. The product owner has a work plan that includes budget estimates and identifies funding sources
5. The product owner has a strong relationship with the contracting officer

Key Questions

1. Who is the product owner?
2. What organizational changes have been made to ensure the product owner has sufficient authority over and support for the project?
3. What does it take for the product owner to add or remove a feature from the service?

Checklist

1. Product Owner identified: Agile ADPQ Product Manager (Margreta)
2. Project Charter identified the owner and was approved by the CEO.
3. Product Owner has a PMP and work experience with IT projects/products.
4. Project Charter included work plan and funding.
5. Not Applicable

Key Questions

1. Margreta(Agile ADPQ Product Manager)
2. Approval by the CEO (with other senior management support) created the organizational change needed.
3. The Product owner participates in Design Review meetings and can add or remove features. See [Meeting notes](#).



PLAY 7 Bring in experienced teams Leonid Marushevskiy

Checklist

1. Member(s) of the team have experience building popular, high-traffic digital services
2. Member(s) of the team have experience designing mobile and web applications
3. Member(s) of the team have experience using automated testing frameworks
4. Member(s) of the team have experience with modern development and operations (DevOps) techniques like continuous integration and continuous deployment
5. Member(s) of the team have experience securing digital services
6. A Federal contracting officer is on the internal team if a third party will be used for development work
7. A Federal budget officer is on the internal team or is a partner
8. The appropriate privacy, civil liberties, and/or legal advisor for the department or agency is a partner

Checklist

1. EngagePoint has experienced and talented engineers with broad experience in development of complex, high-load applications in different domain areas like state government, health care, insurance, etc.
2. EngagePoint has a highly-skilled user experience design group with extensive experience in mobile and web application design.
3. EngagePoint, for more than 5 years, actively uses automated testing in all projects. Our engineers actively used automated testing and test driven development in daily work. We are using JUnit for Java unit testing, Spring Testing Framework for integration tests, Karma JS for JavaScript unit tests, Cucumber for acceptance tests, and Gatling for performance testing.
4. EngagePoint has experienced DevOps engineers that are specialized in continuous integration and continuous deployment using modern tools like Jenkins, Docker, Chef, etc.
5. In our team, we have a Security Analyst that supervises all questions related to security. See team structure in ReadMe.md file.
6. Not applicable for current project
7. Not applicable for current project
8. Not applicable for current project

Checklist



PLAY 8 Choose a modern technology stack Leonid Marushevskiy

Checklist

1. Choose software frameworks that are commonly used by private-sector companies creating similar services
2. Whenever possible, ensure that software can be deployed on a variety of commodity hardware types
3. Ensure that each project has clear, understandable instructions for setting up a local development environment, and that team members can be quickly added or removed from projects
4. Consider open source software solutions at every layer of the stack

Key Questions

1. What is your development stack and why did you choose it?
2. Which databases are you using and why did you choose them?
3. How long does it take for a new team member to start developing?

1. EngagePoint used the JVM platform for software development and Java 8 as the development language. For the implementation of the application prototype, we have used HTML5, CSS3, Bootstrap, and AngularJS. For back-end development, we have used Spring Boot, PostgreSQL, Elasticsearch, Hibernate, Spring Framework, etc. All of these technologies are very popular and commonly used in product systems of different scale.
2. We are using Spring Boot project for our application prototype which provides a variety of deployment options out of the box, such as:
 - Deployment to application containers like Websphere, JBoss, Weblogic, Tomcat, etc. using war file
 - Embedded into application: package container Tomcat or Jetty. In this way, the application can be started on any environment where JRE is installed. This option simplifies deployment to cloud environments.Spring Boot provides extensive configuration capabilities, which help during automated deployment. EngagePoint chooses to deliver software as Docker container, which enables compatibility with virtually any environment, independent of hardware types.
3. EngagePoint used the open source code generator JHipster which provides documentation related to development environment installation as well as other development related instructions. This allows the new team member to start developing quickly by reducing the necessity to learn custom code.
4. Every layer of the stack consists of open source solutions. See Technical approach section 4.1

Key Questions

1. EngagePoint selected the Java Virtual Machine (JVM) platform and Java 1.8 as the prototype's programming language. The JVM platform is commonly used in high-load web applications, such as Google. However, small prototype web applications can also be created quickly and then transitioned to production usage on the same technology platform without having to entirely rewrite the application.

Prototype development using the JVM platform is the correct choice due to the availability of qualified developers, low development and support cost, and low risk in the short- and long-term.

EngagePoint used the open source code generator [JHipster](#), which let us generate a generic application based on [Spring Boot](#), [AngularJS](#) with incorporated minimal for production usage functionality such as user management, user roles, configuration, and monitoring.

Hipster provides tools that accelerate development and reduce the need for custom coding. Entity Generator supports application prototyping, allowing the Technical Architect to describe the standard Entity Relational Diagram using JDL (a domain specific language). Based on JDL, JHipster generates the boilerplate code needed for simple CRUD operations with these entities: [Liquibase](#) scripts for database objects, Hibernate entities, repository classes, Java REST resources, AngularJS controllers, REST client services, routers, unit tests for Java and JavaScript, and sample administrative UI.

The prototype has two maven profiles: DEV and PROD. The DEV profile is used on the local development environment and incorporates in-memory H2 and Elasticsearch engines. Spring Boot provides an embedded lightweight application container, Tomcat, which runs the prototype. We used [Browsersync](#) and [Spring Boot Devtools](#) for automated reload of front-end and back-end code. These techniques reduce development time to seconds as the developer views updates in real time. The PROD profile builds the prototype's production version, which is optimized for production use.

EngagePoint chooses AngularJS for front-end development because this JavaScript MVC framework is mature and widely used. AngularJS is open source and is one of the most popular projects on GitHub. There are an enormous amount of publicly available components for AngularJs.

For back-end development, EngagePoint selects Spring Boot project, which helps to implement the application based on Spring Framework faster. Spring Framework in conjunction with Spring Boot provides a solid development platform for implementation of applications with unlimited complexity and performance requirements.

2. EngagePoint chooses to use PostgreSQL relational database as main persistence storage. This database is the most advanced open source relational database that [provides features](#) comparable with proprietary databases like Oracle and DB2. PostgreSQL is mature, stable, and widely used in enterprise solutions and web applications. PostgreSQL has good support by development tools like Liquibase, Hibernate.
3. The Stack of technologies chosen for application prototype contains open source technologies that are well documented and commonly used. This fact helps new team members quickly come up to speed and be productive. Entry threshold for chosen technologies is relatively low.

PLAY 9 Deploy in a flexible hosting environment **Dmytro Balyam**

Checklist

1. Resources are provisioned on demand
2. Resources scale based on real-time user demand
3. Resources are provisioned through an API
4. Resources are available in multiple regions
5. We only pay for resources we use
6. Static assets are served through a content delivery network
7. Application is hosted on commodity hardware

Key Questions

1. Where is your service hosted?
2. What hardware does your service use to run?
3. What is the demand or usage pattern for your service?
4. What happens to your service when it experiences a surge in traffic or load?
5. How much capacity is available in your hosting environment?
6. How long does it take you to provision a new resource, like an application server?
7. How have you designed your service to scale based on demand?
8. How are you paying for your hosting infrastructure (e.g., by the minute, hourly, daily, monthly, fixed)?
9. Is your service hosted in multiple regions, availability zones, or data centers?
10. In the event of a catastrophic disaster to a datacenter, how long will it take to have the service operational?
11. What would be the impact of a prolonged downtime window?
12. What data redundancy do you have built into the system, and what would be the impact of a catastrophic data loss?
13. How often do you need to contact a person from your hosting provider to get resources or to fix an issue?

Checklist

1. On demand provisioning implemented for 2 cases:
 - a. Docker-based environment (DEV)
In this case additional Application's instances can be increased or decreased using the Docker-composed `scale=n` command. New instances will be automatically added to running HAproxy instance.
 - b. Amazon ECS based environment (QA/PROD)
In this case additional EC2 instances can be added to the application cluster using CLI or AWS Management Console.
2. Auto Scaling Groups are used to leverage containers based on configurable monitoring parameters. Currently implemented rules configured to add 1 instance if CPU load is over 90% more than 10 minutes and remove 1 instance if CPU load below 20% the same amount of time.
3. In both cases resources provisioned by API:
 - a. [Docker](#)
 - b. [Amazon Cloud](#)
4. In case of Docker-based configuration, resources can be deployed and migrated in different Data Centers in different regions. In the case of Amazon ECS, this feature is implemented by Regions and Availability zones.
5. In both cases it is possible to run only the amount of Docker containers or EC2 instances needed and pay only for resources used.
6. In this Prototype there is a very low amount of static content, so this feature is not applicable.
7. Because Docker Engine does not require any Vendor-specific OS of hardware, this Application can be hosted on any commodity hardware.

Key Questions

1. Amazon Web Service as legacy EC2 virtual machine with Docker Engine and cluster in Amazon Container Service.
2. The Prototype runs on any Linux or Windows compatible hardware.
3. Not Applicable
4. In case of surge traffic or high load, Zabbix trigger will be fired for Docker based deployment. In case of Amazon ESC, additional instances will be automatically created to balance a traffic and will be automatically removed when conditions return to normal.
5. Amazon EC2/ESC has virtually unlimited capacity.
6. New resources like application server can be provisioned in a matter of several minutes.
7. The Application uses Hazelcast distribution cache and Load balancers to scale service on demand.
8. Amazon provides different payment policies like pay on demand or pay in advance.
9. Application hosted in 2 Regions with 4 Availability zones.
10. The only disaster-sensitive part of the Solution is the database. After Database recovery application, new containers can be deployed in several minutes using CLI.
11. Not Applicable
12. Proper backup and recovery strategy for the Database is the only vital need for the Solution. Multiply simple application instances can be provisioned for load balancing and redundancy.
13. No need to contact Amazon staff to get resources. Containers automatically restart in case of issues.

PLAY 10 Automate testing and deployments **Dmytro Baly****m, Leonid Marushevskiy**

Checklist

1. Create automated tests that verify all user-facing functionality
2. Create unit and integration tests to verify modules and components
3. Run tests automatically as part of the build process
4. Perform deployments automatically with deployment scripts, continuous delivery services, or similar techniques
5. Conduct load and performance tests at regular intervals, including before public launch

Key Questions

1. What percentage of the code base is covered by automated tests?
2. How long does it take to build, test, and deploy a typical bug fix?
3. How long does it take to build, test, and deploy a new feature into production?
4. How frequently are builds created?
5. What test tools are used?
6. Which deployment automation or continuous integration tools are used?
7. What is the estimated maximum number of concurrent users who will want to use the system?
8. How many simultaneous users could the system handle, according to the most recent capacity test?
9. How does the service perform when you exceed the expected target usage volume? Does it degrade gracefully or catastrophically?
10. What is your scaling strategy when demand increases suddenly?

Checklist

1. Our continuous integration model contains several consecutive levels of auto tests to assure high quality in the delivered software. Every next step is started only when a previous step is passed:
 - Build source code
 - Run a series of tests:
 - Unit tests
 - JavaScript unit tests
 - Sonar code analyzer
 - If all previous tests pass - software is deployed to the Development environment to:
 - Test user-facing functionality by a series of behavior driven auto tests (Cucumber)
 - Run performance tests based on Gatling
 - In all tests pass, software is deployed into AWS PROD environment
2. We used the following tools for unit and integration testing:
 - Junit - Java unit tests
 - Spring Boot integration testing tools - Java integration tests
 - Karma JS - JavaScript unit tests
3. We used Jenkins to run tests automatically
4. We used Jenkins for continuous delivery and automated deployment
5. We used [Gatling](#) for load and performance testing

Key Questions

1. We have around 70% code coverage for back-end Java code and around 40% for front-end JavaScript
2. It takes around 30 minutes
3. It takes around 30 minutes
4. Every commit to the master branch starts and automated build and deployment to the DEV environment
5. We used Cucumber, [Gatling](#), [Karma JS](#), Junit
6. We used Jenkins, Sonar, Cucumber+Selenide, Gatling
7. The application can be scaled automatically using Amazon Elastic Load Balancer. When the service exceeds the expected target usage volume, it degrades gracefully.
8. Not applicable for application prototype. We do not have requirements from customer.
9. Not applicable for application prototype.
10. We used a clustered environment based on EC2 Container Service (ECS). For the Prototype, we created an ECS cluster based on Auto Scaling Group (ESG) of EC2 instances with Elastic Load Balancer (ELB). We can configure various rules in ESG to add and remove EC2 instances based on the current workload or defined schedule.



PLAY 11 Manage security and privacy through reusable processes Leonid Marushevskiy

Checklist

1. Contact the appropriate privacy or legal officer of the department or agency to determine whether a System of Records Notice (SORN), Privacy Impact Assessment, or other review should be conducted
2. Determine, in consultation with a records officer, what data is collected and why, how it is used or shared, how it is stored and secured, and how long it is kept
3. Determine, in consultation with a privacy specialist, whether and how users are notified about how personal information is collected and used, including whether a privacy policy is needed and where it should appear, and how users will be notified in the event of a security breach
4. Consider whether the user should be able to access, delete, or remove their information from the service
5. "Pre-certify" the hosting infrastructure used for the project using FedRAMP
6. Use deployment scripts to ensure configuration of production environment remains consistent and controllable

Key Questions

1. Does the service collect personal information from the user? How is the user notified of this collection?
2. Does it collect more information than necessary? Could the data be used in ways an average user wouldn't expect?
3. How does a user access, correct, delete, or remove personal information?
4. Will any of the personal information stored in the system be shared with other services, people, or partners?
5. How and how often is the service tested for security vulnerabilities?
6. How can someone from the public report a security issue?

Checklist

1. Not applicable for current project
2. Not applicable for current project
3. Not applicable for current project
4. Not applicable for current project
5. Not applicable for current project
6. In the application prototype, we are using configuration management where configuration files for all environments are stored in source code repository, Git. During automated deployment, Jenkins pulls configurations from Git and uses it for application configuration. Git provides full control over configuration versioning.

Key Questions

1. Application collects following personal information from users: First Name, Last Name, email address, and case number. In the profile page, other information (optional to provide) includes phone number, home address, date of birth, gender. This information is collected in the password-protected user profile. The user provides his agreement upon providing personal information.
2. The application collects only minimal necessary information.
3. The User can access, modify, or delete personal information via the password-protected user profile.
4. The System does not share any personal information with other services or individuals. However, users may deliberately share some personal information with a case worker in a message. The System has no control of the information user prefers to share with caseworkers in their conversations. All conversations are accessible to a given authorized user / case worker only.
5. Not tested yet.
6. All issues including security issues can be reported to Public JIRA.



PLAY 12 Use data to drive decisions (Monitoring: **Dmytro Balym**, A/B Testing, customer satisfaction: **Pavel Khozhainov**)

Checklist

1. Monitor system-level resource utilization in real time
2. Monitor system performance in real-time (e.g. response time, latency, throughput, and error rates)
3. Ensure monitoring can measure median, 95th percentile, and 98th percentile performance
4. Create automated alerts based on this monitoring
5. Track concurrent users in real-time, and monitor user behaviors in the aggregate to determine how well the service meets user needs
6. Publish metrics internally
7. Publish metrics externally
8. Use an experimentation tool that supports multivariate testing in production

Key Questions

1. What are the key metrics for the service?
2. How have these metrics performed over the life of the service?
3. Which system monitoring tools are in place?
4. What is the targeted average response time for your service? What percent of requests take more than 1 second, 2 seconds, 4 seconds, and 8 seconds?
5. What is the average response time and percentile breakdown (percent of requests taking more than 1s, 2s, 4s, and 8s) for the top 10 transactions?
6. What is the volume of each of your service's top 10 transactions? What is the percentage of transactions started vs. completed?
7. What is your service's monthly uptime target?
8. What is your service's monthly uptime percentage, including scheduled maintenance? Excluding scheduled maintenance?
9. How does your team receive automated alerts when incidents occur?
10. How does your team respond to incidents? What is your post-mortem process?
11. Which tools are in place to measure user behavior?
12. What tools or technologies are used for A/B testing?
13. How do you measure customer satisfaction?

Checklist

1. Continuous monitoring was implemented by the synergy of built-in AWS containers tools for hardware items and Zabbix for application-specific parameters.
2. Real-time system performance counters export by built-in metrics agent into various monitoring systems: Zabbix, Graphite, Spark.
3. All metrics item has count, median, 95 and 99percentlinevalues.
4. This demo uses Zabbix as monitoring platform and key items have triggers that send alerts to the support team.
5. NA
6. Metrics are internally published on the admin portal of the Application.
7. Metrics are externally published either individually or simultaneously to Zabbix, Graphite, Spark.
8. NA

Key Questions

1. Key metrics are:
 - JVM-based parameters
 - HTTP request statistic
 - REST statistic
 - DataSource statistic
2. Every Application's instance automatically registers in Zabbix with unique host and send metrics updates. Zabbix collects it, analyzes it, triggers alarms, etc.
3. [Zabbix](#)
4. Not Applicable
5. Not Applicable
6. Not Applicable
7. Not Applicable
8. Not Applicable
9. Not Applicable
10. Not Applicable
11. Not Applicable
12. Not Applicable
13. User feedback and usability ratings



PLAY 13 Default to open Leonid Marushevskiy

Checklist

1. Offer users a mechanism to report bugs and issues, and be responsive to these reports
2. Provide datasets to the public, in their entirety, through bulk downloads and APIs (application programming interfaces)
3. Ensure that data from the service is explicitly in the public domain, and that rights are waived globally via an international public domain dedication, such as the "Creative Commons Zero" waiver
4. Catalog data in the agency's enterprise data inventory and add any public datasets to the agency's public data listing
5. Ensure that we maintain the rights to all data developed by third parties in a manner that is releasable and reusable at no cost to the public
6. Ensure that we maintain contractual rights to all custom software developed by third parties in a manner that is publishable and reusable at no cost
7. When appropriate, create an API for third parties and internal users to interact with the service directly
8. When appropriate, publish source code of projects or components online
9. When appropriate, share your development process and progress publicly

Key Questions

1. How are you collecting user feedback for bugs and issues?
2. If there is an API, what capabilities does it provide? Who uses it? How is it documented?
3. If the codebase has not been released under an open source license, explain why.
4. What components are made available to the public as open source?
5. What datasets are made available to the public?

Checklist

1. The application includes a generic email account that can be used to communicate bugs / issues.
2. The dataset used in this application was the dataset provided by California.
3. Data set used was provided as part of California's Open Data.
4. Not Applicable
5. Not Applicable
6. Not Applicable
7. Not Applicable
8. Published materials to Amazon Cloud Service.
9. Done as per the California request.

Key Questions

1. We had multiple review sessions with users to obtain feedback and incorporated changes based on that feedback. The bugs and issues were fixed as soon as the QA and user team reported issues.
2. We have not used any specific proprietary APIs. We have used open source software for all our development. Related links for information and documentation is provided as a part of our delivery.
3. Not Applicable. We have used Open Source licenses.
4. All components including source code, design, architecture, documentation as well as prototype application
5. All data sets used by the application