CASE STUDY

STEP-BY-STEP GUIDE TO DEVELOPING HEALTHCARE MOBILE SOLUTIONS

Client Background

With ceaseless data flows, open operating systems, and large on-board computing capacity, mobile has irreversibly transformed healthcare. However, with great power comes great responsibility—building a mobile healthcare application requires a totally different range of requirements and overall solution development approach. From reference architecture and UI design to data security and regulatory compliance, here is a detailed guide to develop a mobile solution for healthcare.

softserve

Reference Architecture

Presentation Layer
Designing for an effective user experience can be critical to the success of an application. Carry out usability studies, surveys, and interviews to understand what users require and expect from the application, and design with these results in mind.

Business Layer
When designing a business logic subsystem, the goal is to minimize the complexity and decouple components by separating responsibilities into different areas of concerns to promote maintainability, reusability, and testability. For example, business processing, business workflow, and business entities all represent different areas of concern. Within each area, the components designed should focus on the specific area and should not include code related to other areas of concern.

Communication
Communication will often involve sensitive data so it is important to design for security. At the same time, designing an effective communication subsystem is also important for reliability, performance, power, and traffic analytics.

Data Access
Designing the application to use a separate data access subsystem is important for maintainability and extensibility. The data access subsystem should be responsible for managing connections with the data source and for executing commands against the data source. Depending on the business entity design, the data access subsystem may have a dependency on business entities; however, the data access subsystem should never be aware of business processes or workflow components.

Data Processing
It is often necessary to transform data to some other form or format—encrypt/decrypt, compress/decompress, parse/generate JSON, etc.

Sensors
Mobile device sensors can be used to measure motion (accelerometer, gyroscope), position (proximity, magnetic field), and various environmental conditions (light, temperature, or humidity).

System Services
Mobile platforms allow access to some system services—make a phone call, send SMS, get or update information in system address book, calendar, and so on.

Monetizing
There are many different options to make money from the mobile application. Consider the following guidelines when designing monetizing:

- Choose the distribution type for your application—business to business (B2B), in-house distribution within the organization, or public market distribution.
- Choose billing model—free, premium (pay for app), trial (upgrade to premium), freemium (pay for in-app items), subscriptions, or monetize the app through targeted advertising.
- To prevent malicious users from redistributing paid content, do not bundle it in the application bundle, Instead, download it from the remote server.
- Verify the purchase state of unlocked content whenever users access it.
- Obfuscate your in-app billing code.

Cross-Cutting
There are functionalities and components that can be found across different mobile application subsystems, such as security, error handling, logging, and other.

Development Options

When it comes to development effort, focus on these key considerations:

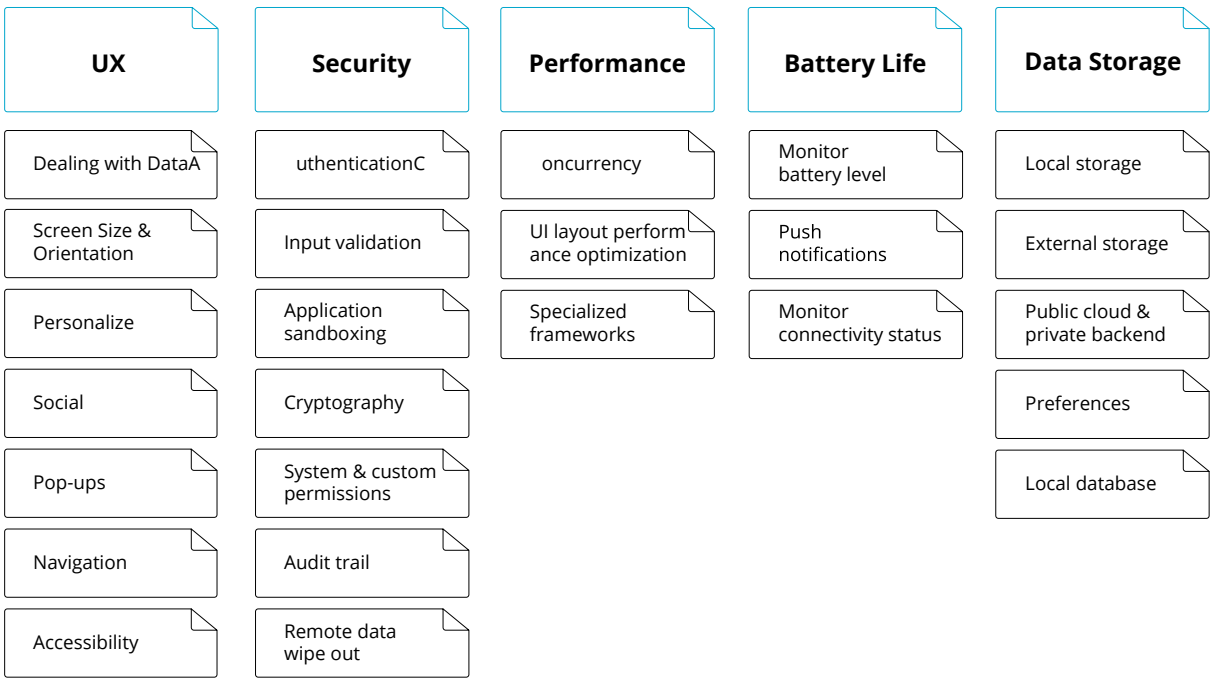
- Development speed
- Available programming expertise
- UX (responsiveness, performance)
- Data persistence (offline) & security
- Access to native device functionality
- Toolset and documentation
- Cross platform support
- Debugging and troubleshooting
- Timely access to OS innovations
- Deployment—distribution and maintenance
- Additional considerations—device fragmentation challenges, monetization, market visibility, etc.

It's important to remember that even though in theory the following frameworks facilitate "build once and deploy on multiple devices," in practice the apps require tweaking for each platform. It is highly recommended to plan on 20-25% additional effort on each platform once the base app is built. For big applications, it can take up to 40-50%. Some device functions may already be supported by the cross-platform frameworks so a careful analysis is needed to determine the additional effort required when developing on multiple platforms. Most cross-platform frameworks let you build plug-ins for device features not supported by the platform itself. However, plug-ins should be developed for each platform, which adds to the overall effort.

Also, mind potential vendor lock-in. Cross-platform frameworks take time to support new functionality that becomes available when a new OS version or device model is released. Note—there is also no guarantee that it will be supported. In contrast, native apps can leverage new capabilities immediately.

Patterns

The following mind map diagram shows some of the most important pattern categories and the actual patterns for these categories.



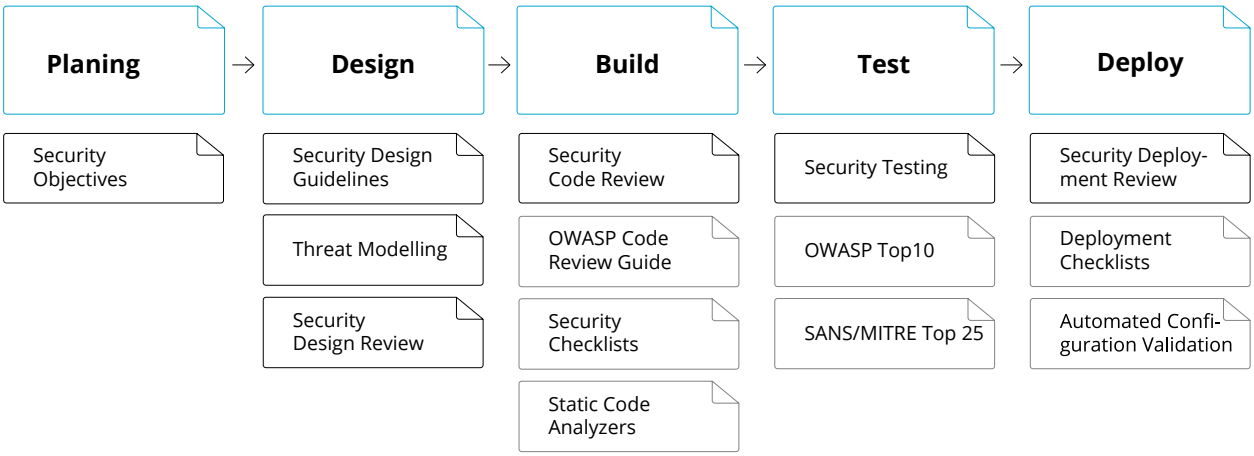
Mobile Application Security

Security is a very important quality attribute of good, large scale, and mission critical mobile software solutions. SoftServe strongly advises that security engineering best practices be embedded within the implementation lifecycle.

According to the National Institute of Standards and Technology (NIST), embedding security into an application produces 30 times the savings as opposed to trying to bolt security on after the application is developed. Other experts put it at 100 times the savings. Insecure applications can lead to security breaches. The practice shows that up to 90% of vulnerabilities could be determined during code and design review against the top 10 common vulnerabilities, therefore, SoftServe includes regular security code review in the development process.

Taking into consideration the possibility of future exposure for the designed solution and growing user base, SoftServe recommends effective security engineering practices, built in conformance with industry recognized and widely adopted Open Web Application Security Project (OWASP) organization system wide and specific recommendations from Microsoft for application security.

The following diagram depicts SoftServe's security engineering practices throughout ASDLC.



Incorporating security into Agile development, allows for the creation of applications that are secure by design—not by chance or circumstance. The proposed solution provides a range of benefits:

1. Effort benefit—the effort to fix the vulnerabilities in the early stage of the system development lifecycle (SDLC) process is much less than the later stage of the process. Once the code is complete and the flaw has not been identified, it is a very tedious and time-consuming process to find problems once the application is ready to move into production. In addition, last minute fixing may affect the entire functionality of the program and hamper deadlines set for product release. Also, it may create other security flaws, which is possible with a large and complex code.
2. Cost benefit—cost is directly proportional to effort required. Not only development cost, but vulnerabilities identified in the production environment may involve more costs. Again, it is well worth it as the costs associated with an attack can be much steeper.
3. Compliance—some compliance makes it necessary to do a secure code review before launching the product. Therefore, an organization following complete SDLC has a better chance of being certified.
4. Reputation—secure code review removes most of the security flaws in the earlier phase making it more secure than only black box assessments. Therefore, there is less chance of the product being compromised, which means less chance of reputation damage.

Data Privacy and Regulatory Compliance

During architecture design and product implementation, it is vitally important to follow HIPAA Technical Safeguards guidelines.

HIPAA Standards	Sections	Implementation
Access Control	164.312(a)(1)	• Unique user identification—UI shall provide authentication mechanism with user name or e-mail and strong password (prohibits registering with weak passwords, previous passwords, and easy guess keywords). • Role-based security—user access level should be “minimum necessary” and based on privileges set by account administrator: view; view and modify; view, modify and collaborate. • User credential encryption—SSL 2048 should be used to protect transferring user credentials over network. The software shall not store user passwords, but only hash codes protected with algorithms approved by NIST (for example, SHA 256+). • Remember me—to be turned off. • Auto-logoff—although this implementation is optional in HIPAA, most healthcare solutions implement it. • Emergency access procedure—depends on business requirements, should be considered during initial phase. • Access for non-registered users—consider masking patient code and limiting (again “minimum necessary” rule) patient personal information shared with such users.

HIPAA Standards	Sections	Implementation
Access Control	164.312(a)(1)	• Patient data encryption—this implementation is optional by HIPAA, but the HITECH Act encourages encrypting patient data. Otherwise, if data is stolen, the new requirements specify that organizations must contact every affected client and report the breach to the government. There are several approaches to meet this—DB encryption, file-system encryption, disk or tape encryption, and patient de-identification. All PHI data stored locally (local storage, cookies, and others) has to be encrypted. Means allowing erase data residing at the client side remotely should be implemented whenever possible (for cases of device loss and similar). • Security testing—SQL injection and other common attack types on data must be prevented at development time. Web application has to be protected from cross-site request forgery (CSRF) attacks. The applications need to go through the security audits, vulnerability, and penetration testing before production. • Audit trail—the software shall be able to track which person accessed which record (down to the patient level) on what date, and whether it was viewed, shared, updated, or deleted. Audit logs must either be free from PHI or be secured—encrypted or stored well protected.

HIPAA Standards	Sections	Implementation
Audit Controls	164.312(b)	• Versioning—modification changes on patient unstructured data such as radiology images, DNA sequence files, etc. should be made on a new copy of the data, preserving all previous data versions. The system shall not delete PHI records physically from the storage. • Patient data retention—recommended retention period of radiological records is six years after the conclusion of treatment. • The minimum retention period of molecular pathology electronic reports, worksheets, and images is 20 years.
Integrity	164.312(c)(1)	• Data authentication—signature or check sum can be used to make sure that data is authentic. Some storage solutions already implement check sum integrity check. • Backups—the data storage solution must sustain device failures and have the ability to quickly detect and repair any lost redundancy. This requirement is critical for a data center or hosting provider. OLTP storage also must provide failover capabilities with backup plans. All backups have to be stored in an encrypted state.
Person or Entity Authentication	164.312(d)	• Unique user identification—see Access Control section. • Authentication with third party software—recommend use of x.509 certificates and tokens as secure mechanisms to authenticate t he connected party.

HIPAA Standards	Sections	Implementation
Transmission Security	164.312(e)(1)	• Integrity Controls—usage of SSL/TLS protocol over HTTP (i.e. HTTPS) or TCP will ensure that data is not modified by "man in the middle." • Encryption—all traffic to and from the hosted software should be protected by SSL 2048. Web Client must not pass any PHI data in URL parameters when sending request to the server.

UI Design

Product success is measured not only by how well the product functions are designed, but also by how well the product serves the customers and the value they perceive from their experience. A great user experience ensures a high-level of user satisfaction and loyalty. With effective UI design, SoftServe can help businesses reduce development time, training and support costs, while also increasing the ROI and customer retention rate.

ABOUT US

SoftServe is a digital authority that advises and provides at the cutting-edge of technology. We reveal, transform, accelerate, and optimize the way enterprises and software companies do business. With expertise across healthcare, retail, media, financial services, software, and more, we implement end-to-end solutions to deliver the innovation, quality, and speed that our clients' users expect.

SoftServe delivers open innovation—from generating compelling new ideas, to developing and implementing transformational products and services.

Our work and client experience are built on a foundation of empathetic, human-focused design that ensures continuity from concept to release.

We empower enterprises and software companies to (re)identify differentiation, accelerate solution development, and vigorously compete in today's digital economy—No matter where you are in your journey.

Visit our **website**, **blog**, **Facebook**, **Twitter**, and **LinkedIn** pages.

NORTH AMERICAN HQ

Tel: +1 866 687 3588 (USA)
Tel: +1 647 948 7638 (Canada)

EUROPEAN HQ

Tel: +44 (0) 800 302 943

info@softserveinc.com
www.softserveinc.com

softserve