

**Department of Computer and Control Engineering**

**Faculty of Engineering**

**Tanta University**

**Insight: Smart C2C Online Services Platform with Administrative Features based on Microservices Architecture**

**(Service Provider: Nurseries)**

Supervisor

**Dr. Amr Elkholy**

**Team Members**

1. Adham Tarek Elsawy
2. Ahmed Fayez Abd Almonem
3. Ahmed Nabil El Nawawy
4. Ahmed Mohammed Saleh
5. Amal Ahmed El-Agder
6. Asmaa Abd Al Aziz Mohammad
7. Youssef Sherif Elgendy

**Acknowledgements**

Finishing this project could not have been achieved without the people that were of great support and help.

We like to thank **Dr. Amr Elkholy.** His supervision, support of the team, guidance, academic knowledge and experience, the project would not have become any close to what it is without him.

We like to thank **Eng. Abdelrahman Metwaly** for his great support. His technical and academic knowledge was very helpful to bring the project to reality.

**Abstract**

**With people becoming more engaged daily, on-demand services have brought the world at their fingertips with one tap, whether they are consumers or business owners.**

**This project is a smart consumer to consumer (C2C) services platform in the form of websites and mobile applications serving busy people fulfilling their demands and service providers with their online visibility.**

**For this case study, we cover nurseries' business owners as the service providers.**

**Table of Contents**

**Team members 1**

**Acknowledgment 2**

**Abstract 3**

**Table of Contents 4**

**Chapter 1: Introduction**

**E-commerce**

**Definition**

**E-commerce Types**

**Problem Statement**

**We are the solution**

**SWOT Analysis**

**Business Model**

**Value Proposition  
 Revenue Model**

**Competitive Advantage   
Market Strategy**

**Business Strategy**

**Business Challenges**

Chapter 1

**Introduction**

**Chapter 1 - Introduction**

**E-commerce**

**What is E-commerce**

**E-commerce** involves the use of the Internet, the World Wide Web (Web), and mobile apps and browsers running on mobile devices to transact business. Although the terms Internet and Web are often used interchangeably, they are actually two very different things. The **Internet**is a worldwide network of computer networks, and the **Web**is one of the Internet’s most popular services, providing access to billions of web pages. More formally, e-commerce can be defined as digitally enabled commercial transactions between and among organizations and individuals.

**Types of E-commerce**

There are a number of different types of e-commerce and many different ways to characterize them. For the most part, we distinguish different types of e-commerce by the nature of the market relationship **–** who is selling to whom. Mobile, social, and local e-commerce can be looked at as subsets of these types of e-commerce.

* **Business to Consumer (B2C) E-commerce –** in which online businesses attempt to reach individual consumers.
* **Business to Business (B2B) E-commerce –** in which businesses focus on selling to other Businesses.
* **Consumer to Consumer (C2C) E-commerce –** in which an online market maker (also called a platform provider) provides a way for consumers to sell to each other. In C2C e-commerce, the consumer prepares the product for market, places the product for auction or sale, and relies on the market maker to provide catalog, search engine, and transaction-clearing capabilities so that products can be easily displayed, discovered, and paid for.
* **Mobile e-commerce (m-commerce)** **–** refers to the use of mobile devices to enable online transactions.
* **Social E-commerce –** a form of e-commerce that is enabled by social networks and online social relationships.
* **Local E-commerce –** a form of e-commerce that is focused on engaging the consumer based on his or her current geographic location.

Our project follows the Consumer to Consumer (C2C) E-commerce type where we play the role of the online market maker.

**Problem Statement**

**Parents’ Problem Statement**

Busy parents find difficulty in searching, reserving, and tracking as they want to find the most suitable nurseries, courses, and activities for their children at the best price. After determining a good nursery, they wish to get multiple choices for reserving. They also want an accessible method to follow up with their children.

**Service Providers’ Problem Statement**

As everything becomes digitized nowadays, it is hard for traditional nurseries’ owners to announce their nurseries compared to smart nurseries that use new technologies for marketing. It was also difficult to clarify all information about their nurseries in detail for all parents one at a time.

**We Are the Solution**

**Parents’ solutions**

Our platform makes it ubiquitous for parents to know all information about more than one nursery from home and determine which courses and activities they need for their children. It also makes it easier to reserve online and follow up on their child's progress.

**Service Providers’ solution**

We help the nurseries’ owners advertise their nurseries, present all detailed information about courses and activities, and display images that help gain parents' trust.

**SWOT Analysis**

We conducted a simple SWOT analysis for strategizing about the business and understanding where we should focus our efforts. In a SWOT analysis the business’s strengths, weaknesses, opportunities, and threats(SWOT) are described.

Strengths **–**

Cost leadership + Easy navigation and transparent pricing + Easy to use platforms

Weaknesses –

Limited financial resources + no prior online experience + no existing user base

Opportunities –

Underserved market + Large scale of target audience + Everything getting digitized nowadays

Threats –

Gaining consumers credibility + Economic downturns + Changing taxes regulations

**Business Model**

A **business model** is a set of planned activities designed to result in a profit in a marketplace.

**Key Elements of a Business Model**

**Value Proposition –** defines how a company’s product or service fulfills the needs of customers.

Our Value Proposition for Parents

* Save time and effort - having all nurseries in one place
* Reduction of price discovery costs – easily find a nursery that define one's needs
* Discover and find nurseries on demand for free
* Price transparency – the ability to know all the prices at any point of time
* Facilitate child progress tracking – using a powerful dashboard

Our Value Proposition for Service Providers

* Online visibility
* Simple Payment Processing and Accounting
* Eliminate internet marketing responsibility
* Ubiquity – available everywhere: at work, at home, and elsewhere on mobile and web

**Revenue Model** – describes how the business will earn revenue, generate profits, and produce a superior return on invested capital.

Our revenue model is a **paid value proposition** based on either a **subscription revenue model** where the service provider is charged a subscription fee for access to the offerings or an **affiliate revenue model** where we steer business to an affiliate and receive a referral fee or percentage of the revenue from any resulting reservations.

**Competitive Advantage –** is what makes a business's services superior compared to all of the competitors.

Our business has neither direct nor indirect online competitors so we benefit from the **first-mover advantage** that results from being the first into a marketplace with a serviceable product. Another competitive advantage is **differentiation** by delivering the service more quicklier than traditional nursery searching methods and having a product that reaches customers better.

**Market Strategy –** defines the plan you put together that details exactly how you intend to enter a new market and attract new customers.

Our market strategy includes

* Integration offline marketing
* Content marketing
* Social media marketing
* Search engine marketing

**Business Strategy**

A **business strategy i**s a set of plans for achieving superior long-term returns on the capital invested in a business firm. A business strategy is therefore a plan for making profits in a competitive environment over the long term.

Our **business strategy** includes

* **Pricing strategy -** depends on keeping the subscription prices low to attract more service providers
* **Customer intimacy -** collecting customer feedback, providing a convenience service and Using testimonials

**Business Challenges**

* **Gaining trust of consumers**
* **Securing funding and limited marketing budget**
* **Keeping up with licensing requirements, market transformations and regulations**
* **No prior management experience**
* **Maintaining Reputation**
* **Balancing quality and growth**
* **Exploding data**

**Chapter 2 - Technologies**

**Introduction**

This chapter takes about the techs used in the application and why these technologies were the choice for our application.

**Back-End Technologies**

**PHP**

**What is PHP?**

PHP is an open source server side scripting language used in web development to create dynamic websites and applications. A survey made by W3Tech demonstrates that almost 79% of the website applications in their data are developed using PHP

**Why is PHP?**

* Open source
* It’s versatile (Platform Independent)
* Strong community support

**Laravel**

**What is Laravel?**

Laravel is a MVC web application framework with elegant, expressive syntax. It frees the developer to create with getting into the foundation details.

**What is MVC framework?**

MVC is short for Model-View-Controller. It is an architectural pattern that separates the application into three logical components. The model, the view, and the controller. Each of the components are built to deal with a specific development concept of an application.

* **MVC Components:**
  + **Model**

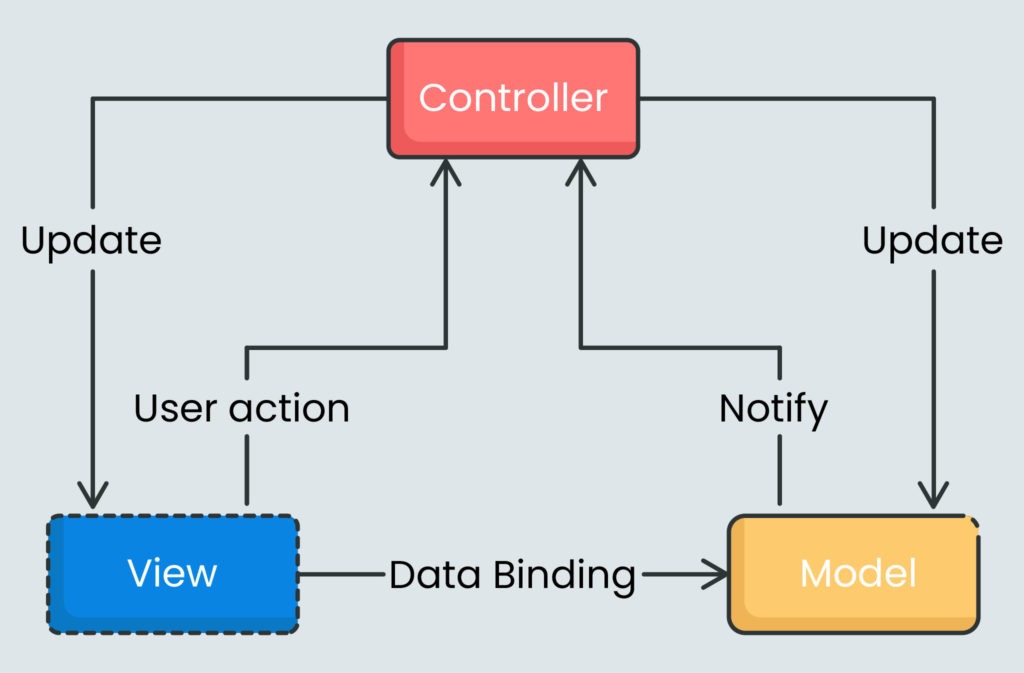
The model component is responsible for all the data related logic that the user works with.

* + **View**

It is used for all the UI logic of the application.

* + **Controller**

The controller act as an interface between the model and the view to process all business logic and incoming requests and also manipulate data with the use of model component and interact with views to help render the final output



**Why is Laravel?**

* MVC Architecture for good performance and support
* Artisan CLI for very simple and fast commands
* Eloquent ORM for easy interaction with the database
* Appropriate App Testing

**Firebase**

**What is Firebase?**

Firebase is a Backend-as-a-Service (Baas). It offers a wide range of tools and services to developers to aid in the creation of high-quality apps. It is built using Google's infrastructure.

Firebase is defined as a NoSQL database program, which stores data in JSON-form documents.

**Firebase Features**

* **Notifications**

With Firebase, notifications can be sent without any additional coding.

* **Firebase cloud messaging**

What is Cloud messaging for Firebase is a Google cloud service that is available for free and enables app developers to deliver messages and notifications to consumers on a wide variety of platforms, including Android, IOS, and web applications.

**Microservice**

**Microservice Architecture**

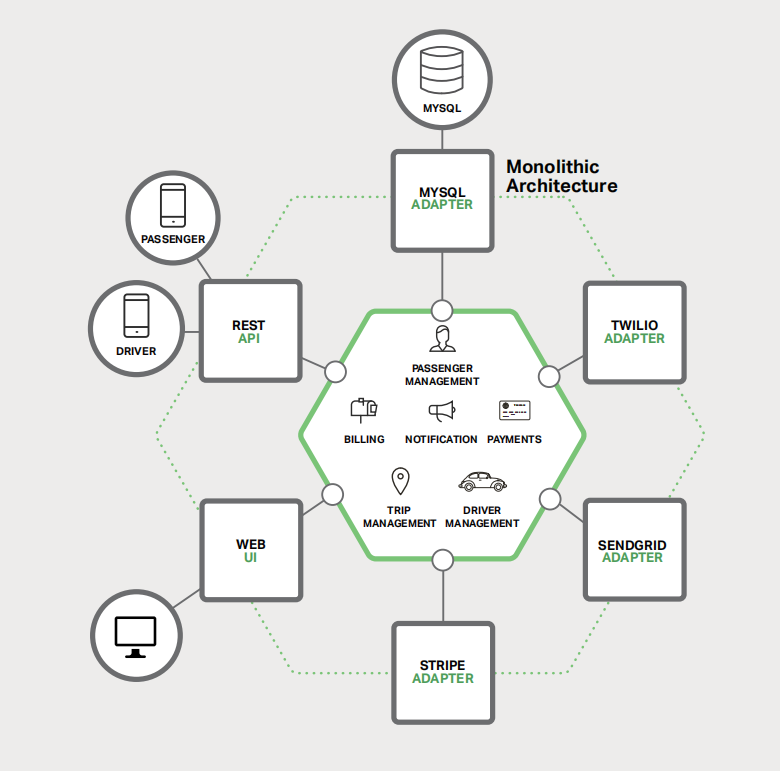
Microservice architecture is an architectural style that structures an application as a collection of services that are highly maintainable and testable, loosely coupled, independently deployable, organized around business capabilities and owned by a small team.

With microservices, an application is developed or refactored, into separate services that “speak” to one another in a well-defined way via APIs, for instance. Each microservice is self-contained, each maintains its own data store, and each can be updated independently of others.

Moving to a microservices-based approach makes app development faster and easier to manage, requiring fewer people to implement more new features. Changes can be made and deployed faster and easier. An application designed as a collection of microservices is easier to run on multiple servers with load balancing, making it easy to handle demand spikes and steady increases in demand over time, while reducing downtime caused by hardware or software problems.

Microservices are a critical part of several significant advancements that are changing the nature of how we work. Agile software development techniques, moving applications to the cloud, DevOps culture, continuous integration and continuous deployment (CI/CD), and the use of containers are all being used alongside microservices to revolutionize application development and delivery.

**The Traditional Way (Monolithic Applications)**

Monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means "composed all in one piece."

Monolithic Architecture Example from NGINX eBook

**Advantages of Monolithic Applications**

* They are simple to develop since our IDEs and other tools are focused on building a single application.
* These kinds of applications are also simple to test. We can implement end-to-end testing by simply launching the application and testing the UI with a testing package such as Selenium.
* Monolithic applications are also simple to deploy. We just copy the packaged application to a server.
* We can also scale the application by running multiple copies behind a load balancer.

In the early stages of the project, it works well. Unfortunately, this simple approach has a huge limitation. Successful applications have a habit of growing over time and eventually becoming huge.

**Disadvantages of Monolithic Applications**

* Once the application has become a large, complex monolith, any attempts at agile development and delivery will flounder.
* One major problem is that the application is overwhelmingly complex. It’s simply too large for any single developer to fully understand. As a result, fixing bugs and implementing new features correctly becomes difficult and time consuming. If the code base is difficult to understand, then changes won’t be made correctly.
* Another problem with a large, complex monolithic application is that it is an obstacle to continuous deployment. Today, the state of the art for SaaS applications is to push changes into production many times a day. This is extremely difficult to do with a complex monolith, since we must redeploy the entire application in order to update any one part of it.
* Another problem with monolithic applications is reliability. Because all modules are running within the same process, a bug in any module, such as a memory leak, can potentially bring down the entire process.
* Moreover, since all instances of the application are identical, that bug will impact the availability of the entire application.
* Finally, it is extremely difficult to adopt new frameworks and languages. For example, let’s imagine that we are using the XYZ framework. It would be extremely expensive to rewrite the entire application to use the newer ABC framework. We are stuck with whatever technology choices we made at the start of the project.

**Microservices**

Diagram

Description automatically generatedMany organizations, such as Amazon, eBay, and Netflix, have solved the previous problems by adopting what is now known as the Microservices Architecture Pattern. Instead of building a single monstrous, monolithic application, the idea is to split the application into set of smaller, interconnected services.

Microservices Architecture Example from NGINX eBook

A service typically implements a set of distinct features or functionality, such as order management, customer management, etc. Each microservice is a mini application that has its own architecture consisting of business logic. Some microservices would expose an API that’s consumed by other microservices or by the application’s clients. Other microservices might implement a web UI. At runtime, each instance is often a cloud virtual machine (VM) or a Docker container.

**Database in microservice**

The Microservices Architecture pattern significantly impacts the relationship between the application and the database. Rather than sharing a single database schema with other services, each service has its own database schema. On the one hand, this approach is at odds with the idea of an enterprise-wide data model. Also, it often results in duplication of some data.

However, having a databaseschema per service is essential if we want to benefit from microservices, because it ensures loose coupling.

A picture containing diagram

Description automatically generated

Database per Service Example from NGINX eBook

**Scalability with microservices**

**A picture containing rectangle

Description automatically generated**The Microservices Architecture pattern corresponds to the Y-axis scaling of the Scale Cube, which is a 3D model of scalability from the excellent book *The Art of Scalability*. The other two scaling axes are X-axis scaling, which consists of running multiple identical copies of the application behind a load balancer, and Z-axis scaling (or data partitioning), where an attribute of the request (for example, the primary key of a row or identity of a customer) is used to route the request to a particular server.

Scale Cube from NGINX eBook

Applications typically use the three types of scaling together. Y-axis scaling decomposes the application into microservices. At runtime, X-axis scaling runs multiple instances of each service behind a load balancer for throughput and availability. Some applications might also use Z-axis scaling to partition the services.

**Advantages of microservices**

* It tackles the problem of complexity. It decomposes what would otherwise be a monstrous monolithic application into a set of services. This enforces a level of modularity that in practice is extremely difficult to achieve with a monolithic codebase. Consequently, individual services are much faster to develop, and much easier to understand and maintain.
* This architecture enables each service to be developed independently by a team that is focused on that service. The developers are free to choose whatever technologies make sense, provided that the service honors the API contract. However, this freedom means that developers are no longer obligated to use the possibly obsolete technologies that existed at the start of a new project. When writing a new service, they have the option of using current technology. Moreover, since services are relatively small, it becomes more feasible to rewrite an old service using current technology.
* The Microservices Architecture pattern enables each microservice to be deployed independently. Developers never need to coordinate the deployment of changes that are local to their service. These kinds of changes can be deployed as soon as they have been tested. This makes continuous deployment possible.
* Finally, the Microservices Architecture pattern enables each service to be scaled independently. We can deploy just the number of instances of each service that satisfy its capacity and availability constraints. Moreover, we can use the hardware that best matches a service’s resource requirements. For example, we can deploy a CPU-intensive image processing service on EC2 Compute Optimized instances and deploy an in-memory database service on EC2 Memory-optimized instances.

**The Drawbacks of microservices**

“There are no silver bullets”

Like every other technology, the Microservices architecture pattern has drawbacks.

* One drawback is the name itself. The term microservice places excessive emphasis on service size. While small services are preferable, it’s important to remember that small services are a means to an end, and not the primary goal. The goal of microservices is to sufficiently decompose the application in order to facilitate agile application development and deployment.
* Another major drawback of microservices is the complexity that arises from the fact that a microservices application is a distributed system. Developers need to choose and implement an inter-process communication mechanism based on either messaging or RPC. Moreover, they must also write code to handle partial failure, since the destination of a request might be slow or unavailable. While none of this is rocket science, it’s much more complex than in a monolithic application, where modules invoke one another via language-level method/procedure calls.
* Another challenge with microservices is the partitioned database architecture. Business transactions that update multiple business entities are fairly common. These kinds of transactions are trivial to implement in a monolithic application because there is a single database. In a microservices-based application, however, we need to update multiple databases owned by different services. Using distributed transactions is usually not an option because of the CAP theorem. We end up having to use an eventual consistency-based approach, which is more challenging for developers.
* Testing a microservices application is also much more complex. For example, it is trivial to write a test class that starts up a monolithic web application and tests its REST API. In contrast, a similar test class for a service would need to launch that service and any services that it depends upon, or at least configure stubs for those services. Once again, this is not rocket science, but it’s important to not underestimate the complexity of doing this.
* Another major challenge with the Microservices Architecture pattern is implementing changes that span multiple services. For example, let’s imagine that we are implementing a story that requires changes to services A, B, and C, where A depends upon B and B depends upon C. In a monolithic application we could simply change the corresponding modules, integrate the changes, and deploy them in one go. In contrast, in a Microservices Architecture pattern we need to carefully plan and coordinate the rollout of changes to each of the services. For example, we would need to update service C, followed by service B, and then finally service A. Fortunately, most changes typically impact only one service; multi-service changes that require coordination are relatively rare.
* Deploying a microservices-based application is also much more complex. A monolithic application is simply deployed on a set of identical servers behind a traditional load balancer. Each application instance is configured with the locations (host and ports) of infrastructure services such as the database and a message broker. In contrast, a microservice application typically consists of a large number of services. Each service will have multiple runtime instances. That’s many more moving parts that need to be configured, deployed, scaled, and monitored.
* In addition, we will also need to implement a service discovery mechanism that enables a service to discover the locations (hosts and ports) of any other services it needs to communicate with. Consequently, successfully deploying a microservices application requires greater control of deployment methods by developers and a high level of automation.

**Front-End Technologies**

**Bootstrap**

Bootstrap is the most popular CSS framework for developing responsive and mobile first websites.

**Material UI**

It is an open-source project that features React components that implement Google’s material design.

**Single page application or SPA**

A Single-page Application is a type of web application loaded from one page, and all user interaction with this service is carried out, using one screen (page).

When users launch such an app, dynamic rewriting using JavaScript allows avoiding reloads or additional pages loading. Users see all the main content in the browser, and when scrolling or going to other pages, the necessary elements are simply loaded instead of a full app reload.

Examples of single-page applications include Twitter, Gmail, Evernote, Pinterest, and many other web apps that use the technology of a single-page application to provide flexible and scalable user experiences.

**Advantages**

* Performance
* Improved user experience
* Data caching
* Development speed
* Ease of debugging

**Disadvantages**

* Problems with SEO
* Downloading time
* JavaScript support is necessary

**Our Solutions for these problems**

JavaScript is now supported from all browsers so it is not an issue nowadays and the search engines develop its bots to make able to read the JavaScript files, so almost in a few months SEO will not be a problem for SPAs.

And to overcome the problem of having a low FCP score or (first content full paint) which is the time from when the page starts loading to when any part of the page's content is rendered on the screen, we can make a mix between the three options of making a web site which are:

* static side generation
* server-side rendering
* client-side rendering (SPA)

Save the home page statically on a CDN, render product pages server side and cache them and load user related content client-side. After the page loads you may also use an SPA that uses SSR for the first page load and then continues in SPA mode for the rest of the user visit.

In that case we will have a better FCP score and better SEO and at the same time we will have link clicks and form submissions not triggering a full-page load.

To make an SPA, we built it using React library

**React**

It is a JavaScript library for building user interfaces, it is used to build a single-page application.

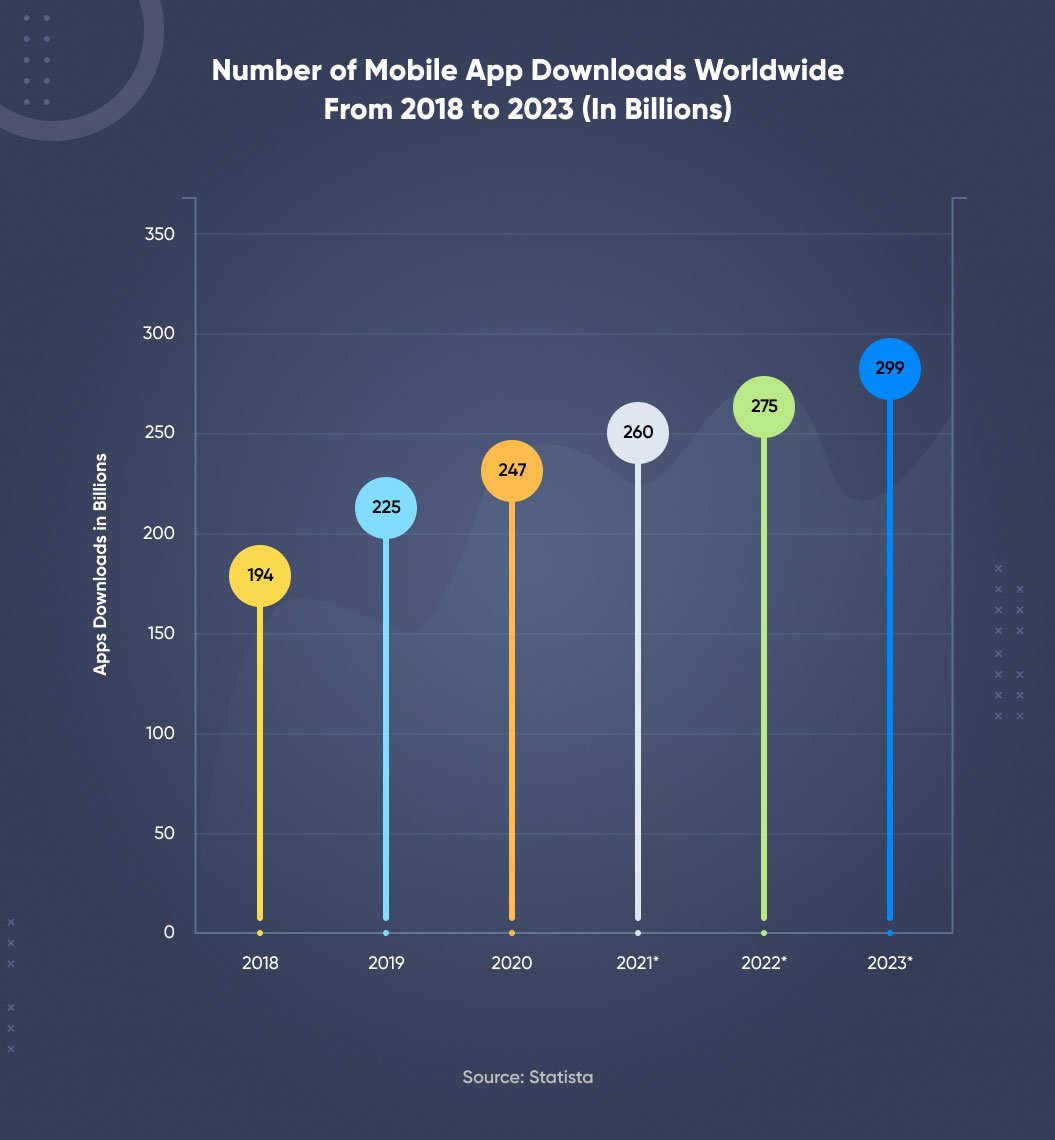
ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front-end library which is responsible only for the view layer of the application. It was initially developed and maintained by Facebook and later used in its products like WhatsApp & Instagram.

**There are other JavaScript libraries that make a SPA like Angular.js and Veu.js but we use React because its community and the plugins that help developers to save their time.**

**Mobile Development**

In today's computing world, different technologies have emerged. These have grown to support the existing computer networks all over the world. With mobile computing, we find that the need to be confined within one physical location has been eradicated.

Mobile applications have become an inseparable part of our lives. From socializing, shopping, travel booking, learning, to watching our favorite movies and TV shows, we have a mobile app for almost everything. Since millions of mobile applications already there on the Google Play Store and Apple’s App Store, it is not easy for a new app to grab the attention of users.

In 2023, it is estimated that the annual number of app downloads worldwide will amount to 299 billion, up from approximately 247 billion global app downloads in 2020.

A mobile app development services provider has to consider a plethora of factors to make an app successful on the Google Play Store or App Store. Choosing the right technology for custom mobile app development is one of them.

**Mobile Technologies**

The two most dominant mobile platforms on the market are Android and iOS, sharing three mobile app development frameworks: Native, Hybrid and Cross-Platform.

**Native**

Native apps are built specifically for a particular operating system leveraging platform-specific programming languages. When it comes to developing native apps for **Android**: Java or Kotlin are used and in the case of **IOS**: Objective-C or Swift is used.

**Examples of Best Native App:** Google Maps, Pinterest, Spotify.

**Hybrid**

Hybrid app development is the combination of native and web solutions where developers need to embed the code written with the languages such as CSS, HTML, and JavaScript into a native app with the help of plugins including Ionic Capacitor, Apache Cordova, and so on which enables to get the access of native functionalities.

**Examples of Best Hybrid Apps:** Instagram, Gmail.

**Cross-Platform:**

Cross-platform development points to the process of creating an app that works on several platforms. Cross-platform applications are popular in today’s times, all credits to React Native, Xamarin, and Flutter frameworks.

**Examples of Best Cross-platform Apps:** Skype, Slack.

We built our project by using cross-platform with flutter.

**Why Cross-Platform?**

Cross-platform frameworks operate on the agenda to develop shareable and reusable code for building apps for different OS. Writing code once and reusing the same on multiple platforms helps in minimizing the development costs and efforts, Cross-platform apps will ensure hassle-free implementation, robust functionality, and affordable production.

Despite using cross-platform, don’t expect high performance and customization with cross-platform app development framework.

**Why Flutter?**

**Advantages**

* Faster development and deployment.
* Quality documentation.
* Feature-rich user interfaces.
* Single Code Base.
* Compatibility with older devices.

**Disadvantages**

* Framework age: Dart and Flutter are pretty young.
* Project size.

**Chapter 3 – System Design**

**Chapter 4 – Client Part**