



# Kotlin

## MOBILE SYSTEMS

### Master in informatics

Leandro Alexandre  
leandro.alexandre@ipb.pt

Original content by Paulo Alves – adapted by Leandro Alexandre



# Syllabus

## 1. Project development methodologies

- Project based learning (**PBL**)
- Collaborative software development using **GitHub**
- Agile software development using **Kanban**
- Requirements engineering
- Software Modeling using **UML**



# Syllabus

## 2. Interface design

- Figma
- Wireframes and Mockups
- Storyboarding
- Material Design
- Bootstrap
- Mockup design
- Prototyping



# Syllabus

## 3. Android Development

- Development technologies
- Frameworks and Toolkits
- Kotlin language
- Layout design
- Activities and intents
- Data bind
- SQLite database



# Syllabus

## 3. Android Development

- Backend REST Services
- Authentication
- Publish to App store



# Syllabus

## Project (**Project Based Learning**)

### Mobile Systems

- Android Application
  - Kotlin
  - REST API
  - JWT



# Syllabus

## Software Architectures

- Backend
  - [Beeceptor](#)
  - [Postman](#)
  - [Dummyjson](#)
  - [Jsonplaceholder](#)



# Syllabus

## Assessment methods

- Individual activities - **30%** (5 activities)
  - Mockup design in Figma
  - Bootstrap implementation
  - Android app implementation
  - Android App with REST Services
  - GitHub tasks and commits





# Syllabus

## Project – **70%**

- Prototype (**30%**)
  - User stories
  - UML models (use cases, activities and classes)
  - Mockups
  - Prototype



# Syllabus

## Final Project (**70%**)

- Project functionalities and quality
- Public Presentation
- Report
- GitHub tasks and commits
- Team collaboration on Slack or Discord
- Participation on meetings



# Syllabus

How the Subject works:

- **Classes**
  - Exercices
  - Work on project
  - Stakeholders meeting (each 15 days)
- **Out of the class**
  - Learning by MOOCs
  - Other resources



# Syllabus

## Important Dates:

- 04/03 - Constitution of groups (proposed by students)
- 04/03 - Team interviews
- 11/03 - Assignment of the project theme
- 11/03 - Team leader election
- 06/05 - Functional Prototype (UML, Figma mockups and prototype)
- 25/06 - Final Project



# Syllabus

Important notes:

Groups (students organize the groups)

- **4 to 6** students

Project team leader (must be chosen by the group)

- Project Coordinator / Scrum Master
- Is responsible to define the tasks
- Coordinate the team
- Report to Stakeholder



# Syllabus

## Software

- GIMP (image editor)
  - <https://www.gimp.org/>
- Figma (Mockups)
  - <https://www.figma.com/education/>
- Android Studio
  - <https://developer.android.com/studio>



# Syllabus

## Report

- Office 365 Education
  - Office
  - Share de Report for collaborative edition
- Optionally
  - Google Docs
  - <https://www.google.com/docs/about/>
  - Overleaf (Latex) <https://www.overleaf.com/>



# Syllabus

## UML modeling

- Visual Paradigm (UML)
  - <https://www.visual-paradigm.com/solution/freeumltool>
  - <https://online.visual-paradigm.com/pt/>





# Syllabus

## Prerequisites

- HTML
  - <https://www.w3schools.com/html/default.asp>
- CSS
  - <https://www.w3schools.com/css/default.asp>
- JavaScript
  - <https://www.w3schools.com/js/default.asp>
- Object Oriented Programming (Python, C#, **Java**, C++)



## Lets know each other?

Go to Forum on Virtual and do a small presentation of yourselves:

- Name and Age
- Where are you from
- Area of interest (web, mobile, sysadmin, security, AI, etc)
- Favorite programming language
- Hobbies
- ...



# Project development methodologies

## 1.1 Project Based Learning

What is Project Based Learning (PBL)?

- PBL is a teaching method in which students gain knowledge

and skills by working for an extended period of time to investigate and respond to an authentic, engaging and complex question, problem, or challenge.



# Project development methodologies

Essential Project Design Elements:

- Key Knowledge, Understanding, and Success Skills
- Challenging Problem or Question
- Sustained Inquiry
- Authenticity



# Project development methodologies

- Student Voice & Choice
- Reflection
- Critique & Revision
- Public Product



# Project development methodologies

Why Project Based Learning (PBL)?

- PBL makes school more engaging for students.
- PBL improves learning
- PBL builds success skills for college, career, and life
- PBL helps address standards
- PBL provides opportunities for students to use technology

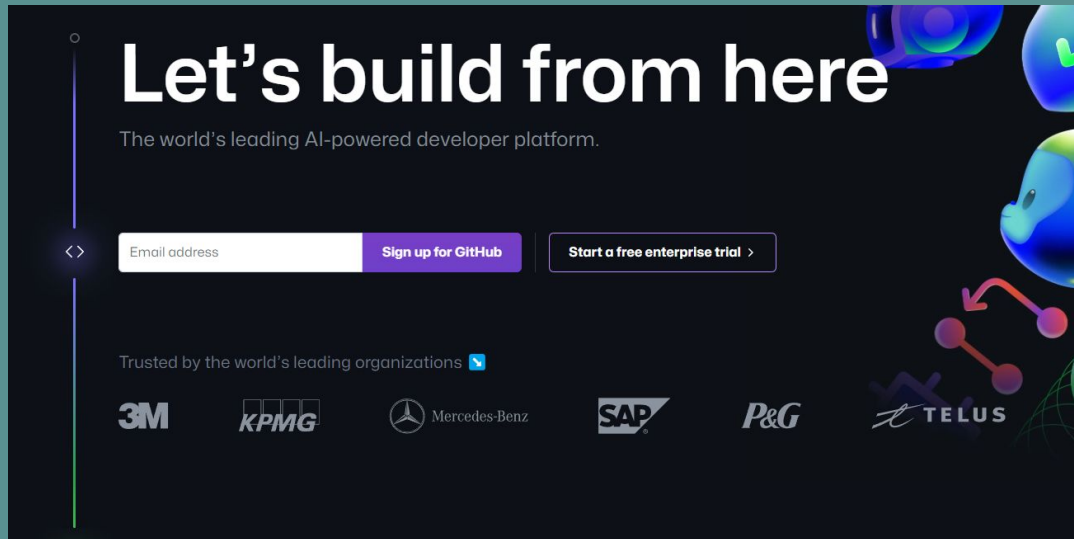


# Project development methodologies

- PBL makes teaching more enjoyable and rewarding
- PBL connects students and schools with communities and the real world
- PBL promotes educational equity

# Project development methodologies

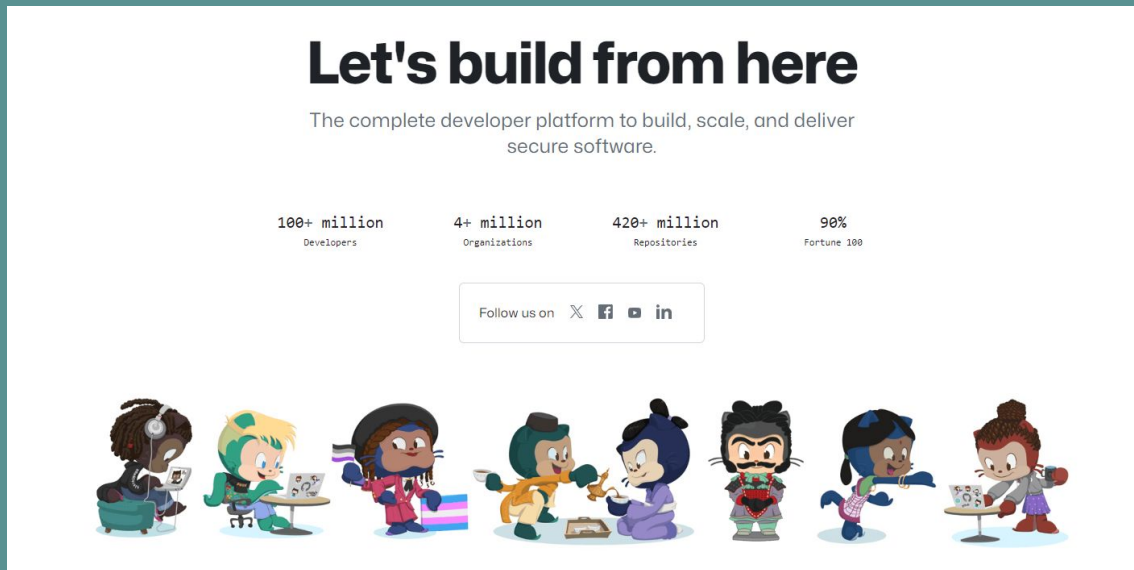
## 1.2 Collaborative software development using GitHub





# Project development methodologies

## 1.2 Collaborative software development using GitHub




**Let's build from here**

The complete developer platform to build, scale, and deliver secure software.

100+ million Developers	4+ million Organizations	420+ million Repositories	90% Fortune 100
----------------------------	-----------------------------	------------------------------	--------------------

Follow us on [X](#) [f](#) [v](#) [in](#)



A row of seven diverse cartoon characters representing developers. From left to right: a Black man with dreadlocks wearing headphones and holding a smartphone; a person with green hair and a green hoodie sitting at a desk with a laptop; a person with a beard and a red jacket holding a flag with the GitHub Octocat logo; a person with green hair and a green hoodie holding a small object; a person with purple hair and a purple robe holding a small object; a person with a beard and a red jacket holding a small object; and a person with red hair and a red jacket holding a small object.



# Project development methodologies

## 1.3 Agile software development using Kanban

What is Kanban?

- Kanban is a visual system for managing work as it moves through a process.
- Kanban visualizes both the process (the workflow) and the actual work passing through that process.
- The goal of Kanban is to identify potential bottlenecks in your process and fix them so work can flow through it cost-effectively at an optimal speed or throughput.

# Project development methodologies



# Project development methodologies

**OctoArcade Invaders**

Filter by keyword or by field

Planning | Sprint Board | Alpha | Roadmap | My work | Features | Priority | By person | Status Board | By status | By Sprint | Done

### Not Started 19 Estimate: 37

- planning-tracking-demo #810  
Beta go-no-go meeting
- planning-tracking-demo #800  
Save score across levels
- planning-tracking-demo #784  
Interviews with media outlets  
**epic**
- Draft  
Enable for teams
- planning-tracking-demo #1161  
tweak difficulty
- planning-tracking-demo #1167  
Update README.md
- Draft  
Prevent the Konami code from bringing down all of GitHub

### Planning 19 Estimate: 109

- planning-tracking-demo #823  
Updates and bug fixes to engine from Beta  
**bug** **demo**
- planning-tracking-demo #824  
Beta signup page  
**need help**
- planning-tracking-demo #806  
[Tracking] Upsell / Growth experience  
**backlog** **feature**
- planning-tracking-demo #818  
Account subscription design
- planning-tracking-demo #828  
Acquire domain for launch
- planning-tracking-demo #832  
Final creative shots from game
- planning-tracking-demo #829

### Building 8 Estimate: 40

- planning-tracking-demo #1160  
Update documentation
- planning-tracking-demo #814  
Updates to collision logic  
**enhancement**
- planning-tracking-demo #816  
Free and paid levels  
**need help**
- planning-tracking-demo #831  
Documentation and Support  
**need help**
- planning-tracking-demo #821  
Updates to alien, beam, bomb and cannon sprites  
**#370**
- planning-tracking-demo #802  
Updates to velocity of the ship and alien movements

### Review 5 Estimate: 17

- planning-tracking-demo #822  
Hero site - Development  
**#12** **#1160** **in-review** **task**  
**urgent**
- planning-tracking-demo #808  
General bug fixes from Alpha feedback  
**#992**
- planning-tracking-demo #1151  
Design new launch screen  
**#374** **web**
- planning-tracking-demo #793  
Polished alien, beam, and cannon sprite files
- planning-tracking-demo #1101  
[Tracking] Integrate payments system  
**backlog** **feature**

+ Add item



# Project development methodologies

## 1.4 Requirements engineering

- Requirements engineering is the process of defining, documenting, and maintaining requirements in the engineering design process.



# Project development methodologies

It is a four-step process, which includes:

- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation
- Software Requirement Management



# Project development methodologies

## 1.4.1 Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards



# Project development methodologies

## 1.4.2 Requirement Elicitation and Analysis:

This is also known as the gathering of requirements. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.





# Project development methodologies

## 1.4.3 **Software Requirement Specification:**

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.



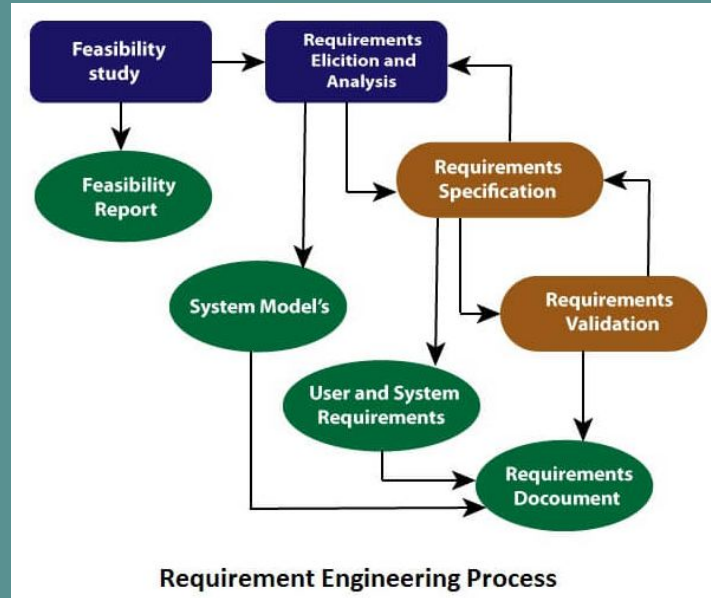
# Project development methodologies

## 1.4.4 **Software Requirement Validation:**

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions:

- If they can practically implement
- If they are correct and as per the functionality and specially of software
- If there are any ambiguities
- If they are full
- If they can describe

# Project development methodologies



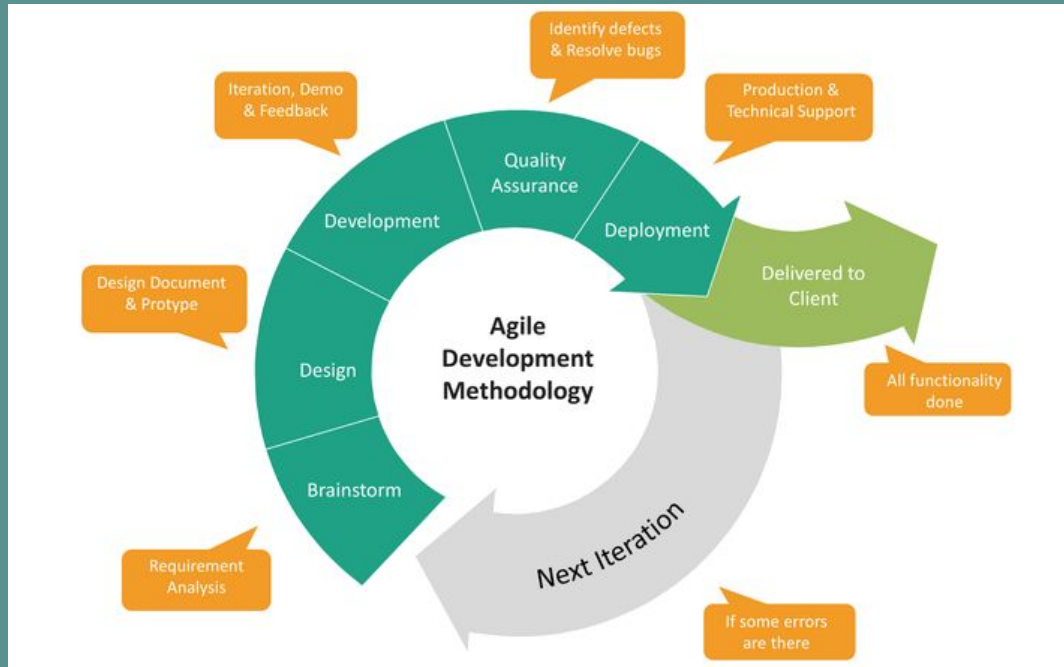
# Project development methodologies

## 6 Phases of the Software Development Life Cycle



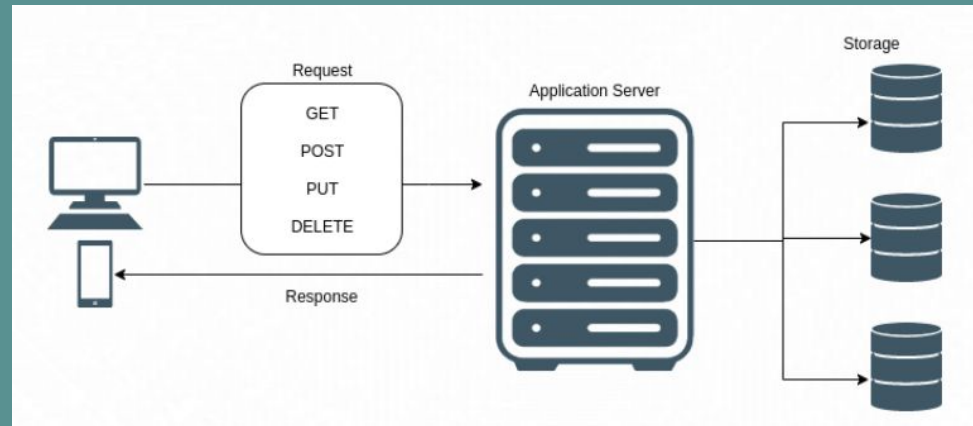
# Project development methodologies

AGILE:



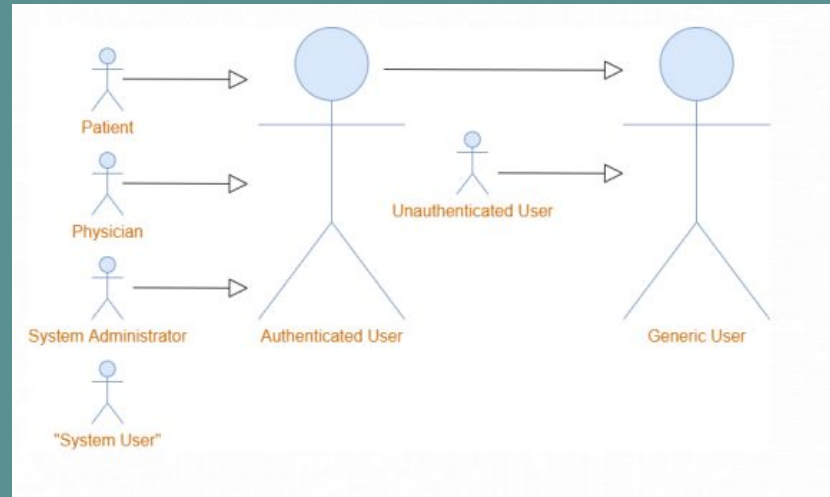
# Project development methodologies

## 1.5. Software Modeling using UML Architecture



# Project development methodologies

## Actors (Visual Paradigm)



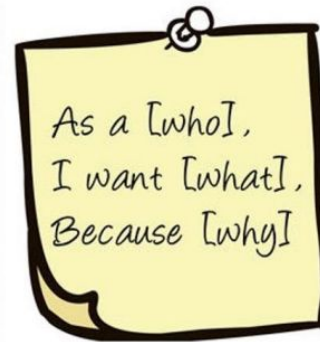
# Project development methodologies

User stories (Office 365)

## Writing a user story

- 1 Define your **end user**
- 2 Specify what **they want**
- 3 Describe **the benefit**
- 4 Add **acceptance criteria**

Copyright © 2019 Knowledge Train Limited





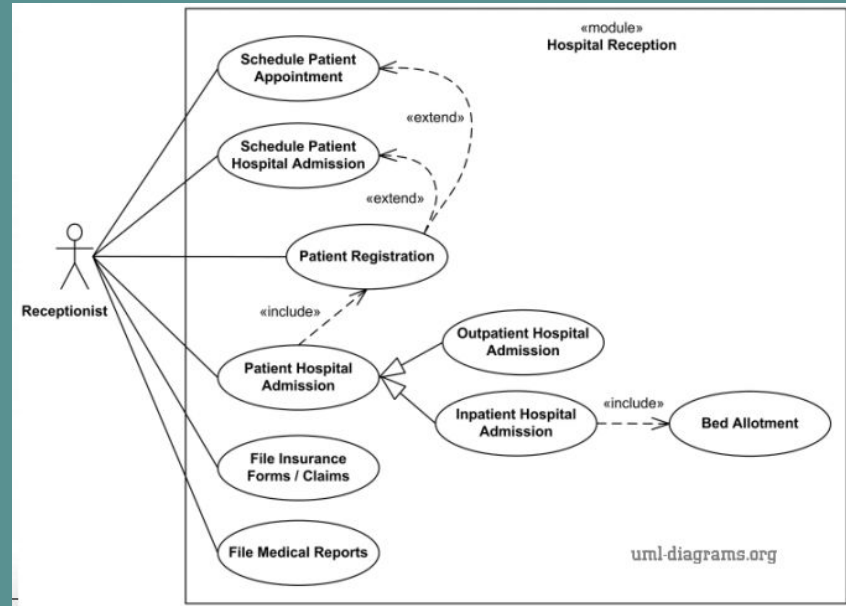
# Project development methodologies

## User stories Example - Patient

Identifier	Name	Priority	Description
USP01	Display Wearable Position	5	As a <b>Patient</b> I want to be able to <b>see where the wearable must be placed</b>
USP02	Display Treatment Plan	3	As a <b>Patient</b> I want to be able to see <b>what my treatment plan is</b>
USP03	Message Physician	1	As a <b>Patient</b> I want to be able to <b>send a message</b> to the physician that is responsible for my monitoring if I think that something is wrong.

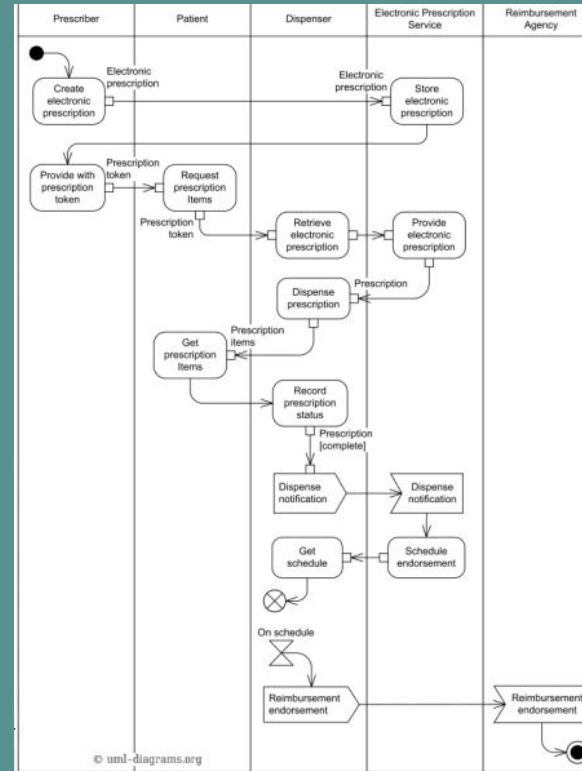
# Project development methodologies

## Use cases (Visual Paradigm)



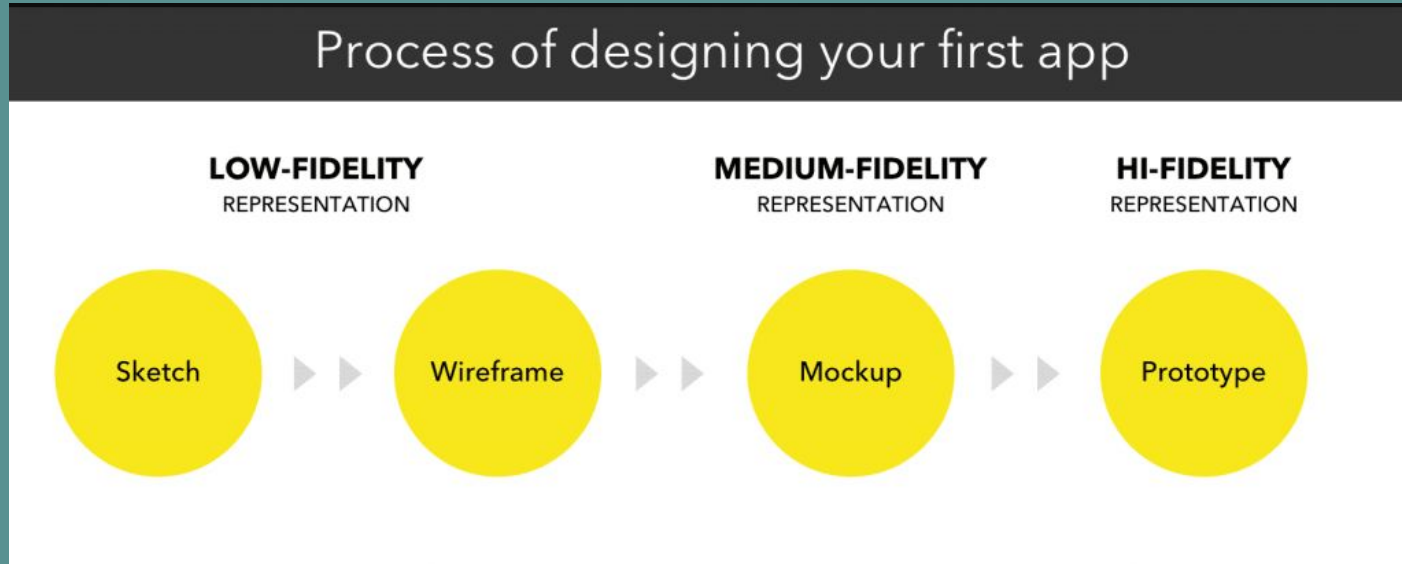
# Project development methodologies

## Activities (Visual Paradigm)



# Interface design

## Interface design





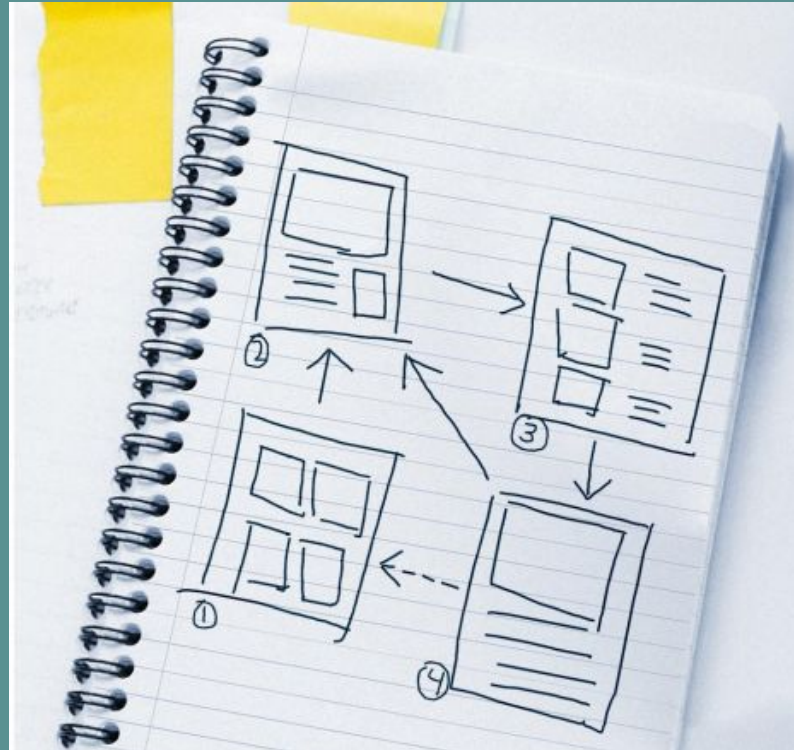
# Interface design

## Sketch

- Raw freehand drawing on a piece of paper
- Gives a low-fidelity representation of the app
- It is the fastest way to get the idea ready for brainstorming

# Interface design

Sketch





# Interface design

## Wireframes

- Equivalent to the skeleton or simple structure of your website/app
- Each one is used to describe the functionality of a product as well as
  - relations between views (what will happen when you click a certain button).
- The decisions on what (content/features) and where to put on the website or app are usually made during this stage
- This step does not cover the product's design



# Interface design

## Mockups

- With this representation, you can start to work on the development process and the developer can make your mockups a reality
- Any mockup will provide a medium-fidelity representation
- Add colours, fonts, text (Lorem ipsum), images, logos and anything else that will shape your wireframe





# Interface design

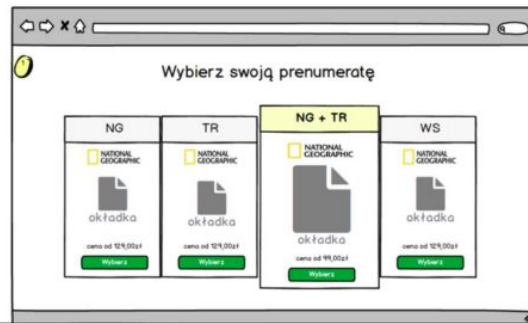
## Mockups

- Your result is a static map of the app
- Think about User Interface Practices while shaping this step
- <https://brainhub.eu/library/wireframe-mockup-prototype>

# Interface design

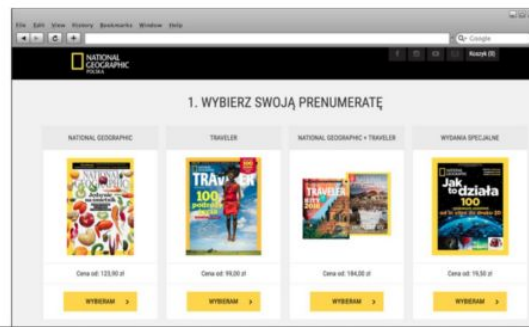
## WIREFRAME

Structure + Functions + Content



## MOCKUP

Style + Colours + Right Content





# Interface design

UX VS UI  
It's Different?



# Interface design

## **UX - User Experience**

- User experience - Means the relationship that a person has with a particular product or service. From the moment the user is interested, comments, researches and buys something, this is all User Experience



# Interface design

## **UX - User Experience**

- UX Designer is concerned with each step in which the user interacts with the product or service.
- It aims to make this interaction occur as smoothly as possible



# Interface design

## **UX - User Experience**

- It is the combination of empathy, usability, technology and the humanization of machines during the development process
- Research, perform usability tests and apply changes to the project, so that the product or service has a positive impact on users, in line with business objectives

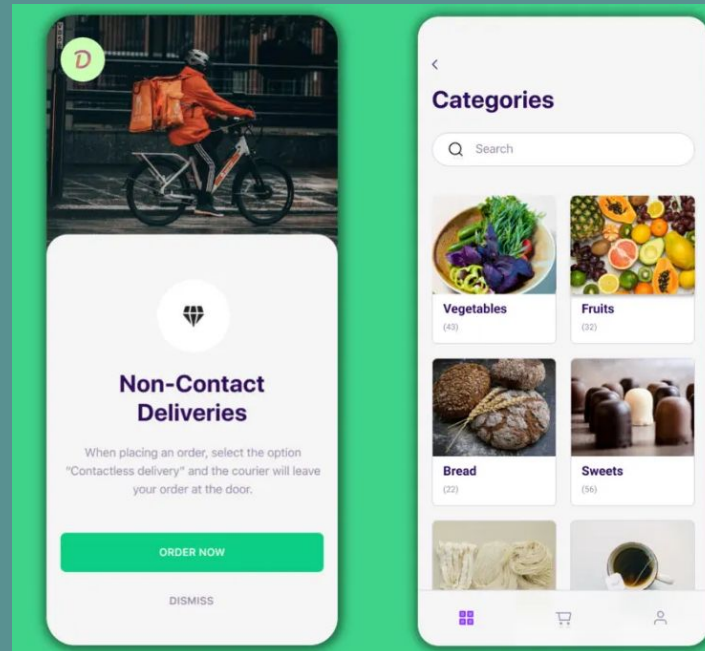


# Interface design

## **UI design - User Interface**

- User Interface - Represents everything that is used when interacting with a product
- UI is just an interface / screen, a way for a person to interact with something
- UI Designer is primarily responsible for creating functional interfaces

# Interface design







# Interface design

## **UX Designer**

- Macrointeractions architect - process of identifying usability problems since the user is faced with the problem until he/she finishes the process he/she wants to execute.

## **UI Designer**

- Microinteractions architect, specific stage of navigation in the application, how the user interacts with it in a harmonious way.



# Interface design

When building a mockup:

- Make the mockup visually attractive
- Keep the mockup consistent
- Make it simple and clear for everyone
- Focus on content
- Continuously test the mockup



# Interface design

What not to do:

- Ignore the target audience and their needs
- Use many fonts (maximum 3 fonts)
- Use many colors (maximum 1 primary and 3 secondary colors)
- Sacrifice usability in favor of design
- Don't use microinteractions and transitions



# Interface design

## Usability

- It is a measure of the effectiveness of using a product / design that a specific user has in a specific context to achieve a defined objective, in an effective, efficient and satisfactory way.
- Designers generally measure the usability of a design throughout the development process - from wireframes to the final product - to ensure maximum usability.
- Usability is the second level in the user experience



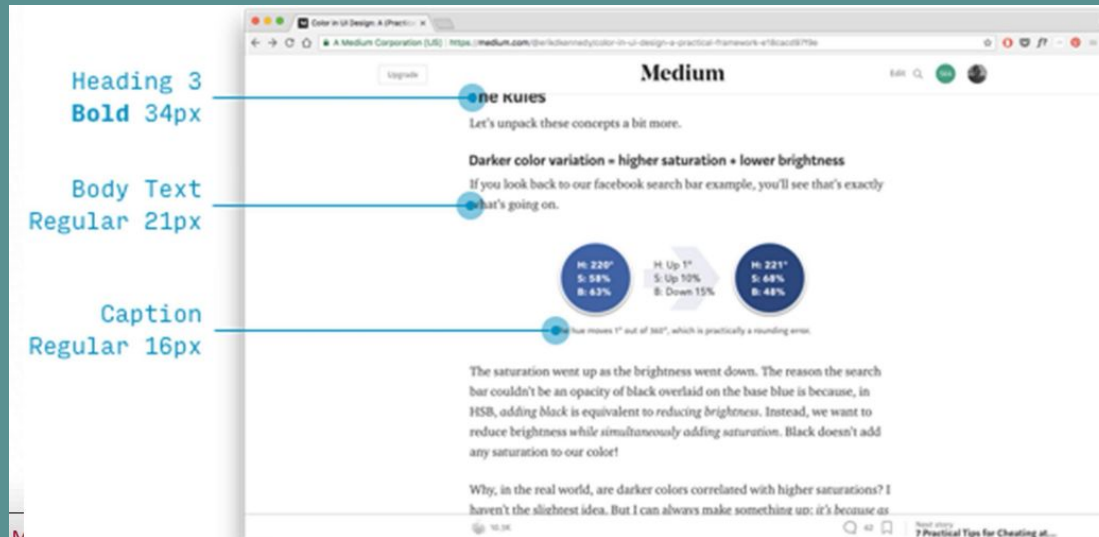
# Interface design

## **Usability principles:**

- Effectiveness - supports users in completing actions accurately.
- Efficiency - users can perform tasks quickly through the easiest process.
- Engagement - users find it enjoyable to use and appropriate for their area / expertise.
- Error tolerance - Supports a variety of user actions and only shows an error in really wrong situations.
- Ease of learning - New users can meet their goals with ease and are even easier on future visits.

# Interface design

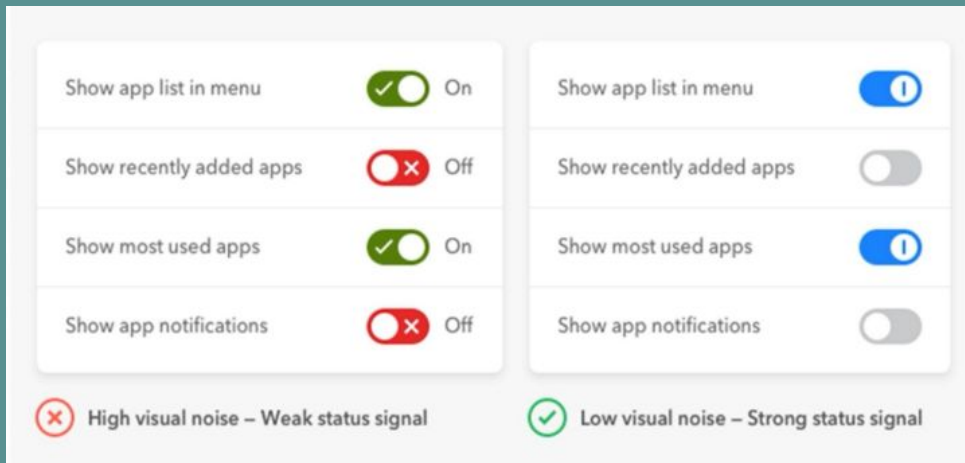
- Use of fonts (maximum 3 fonts)





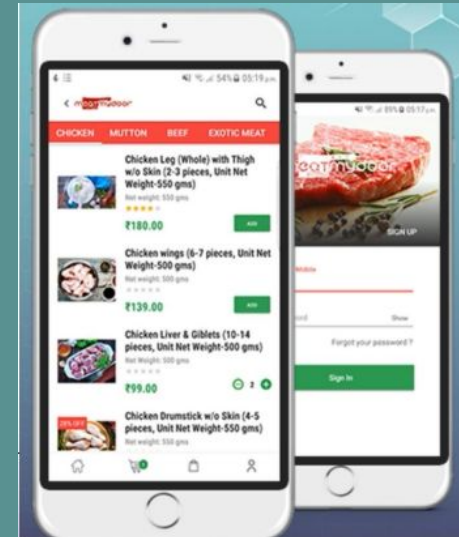
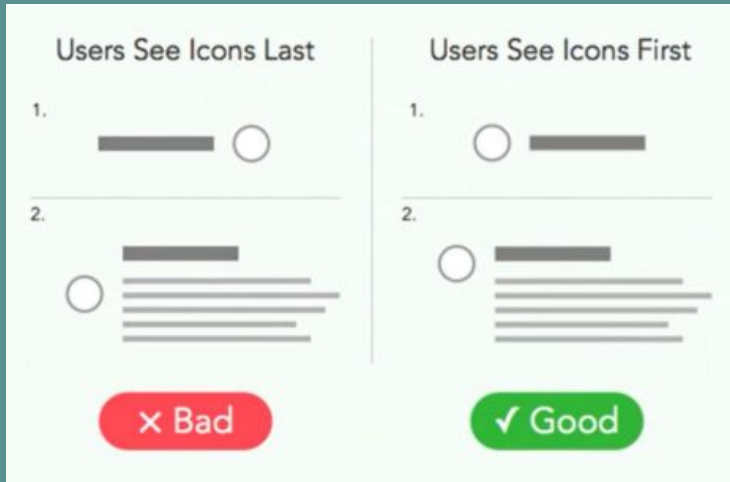
# Interface design

## Use colors in the right way



# Interface design

Users view logos and images first (icons to the left of text and aligned with the first line)





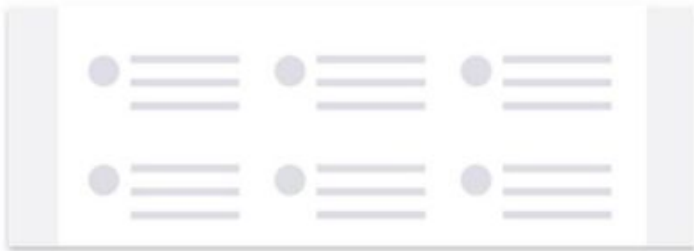
# Interface design

- Space between elements

Do

Visual Perception:

1 section containing 6 groups of 1 dot and 3 lines each.



Don't

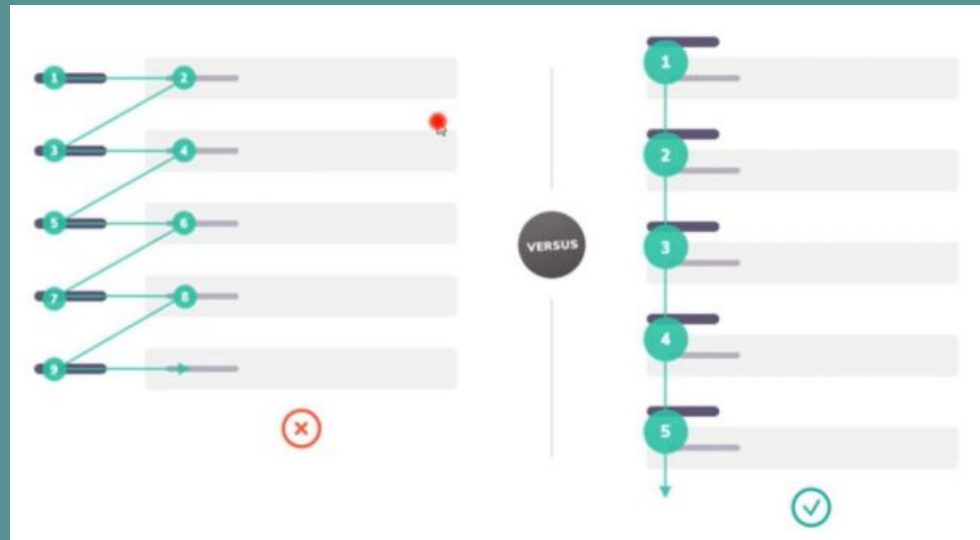
Visual Perception:

1 section with 6 dots and many lines (18) without clear grouping. In this case our brain uses a different law to categorise the elements which is the law of similarity.





# Interface design

- Elements organization



# Interface design

**The system should help the user to avoid mistakes before they happen  
(Suggest succinct but specific information)**

<div></div> <div>Email <input type="text"/></div> <div>Please enter your email address in format: yourname@example.com</div>	<div></div> <div>Email <input type="text"/></div> <div>The email address you entered does not match with required format. Please enter your email address using standard format.</div>
A precise and meaningful indication of the desired behavior.	A long message containing unnecessary details but missing the required information.

# Interface design

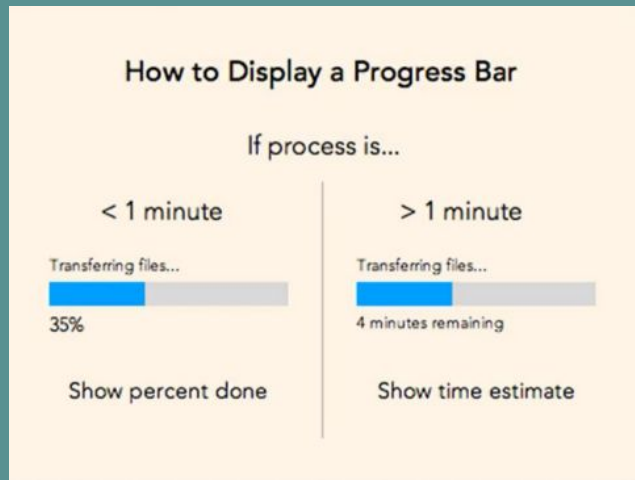
Use steps and keep data

The image displays three overlapping mobile app screens, each representing a step in a registration process. Each screen features a title, a progress indicator, input fields, and a green action button.

- Left Screen (Profil korisnika):** The title is "Profil korisnika" with a user icon. The progress bar shows the first of three steps completed. It contains two text input fields labeled "Ime" and "Prezime", both with masked characters. A green "NASTAVI" button is at the bottom.
- Middle Screen (Pristupni podaci):** The title is "Pristupni podaci" with a lock icon. The progress bar shows the second of three steps completed. It contains three text input fields: "Korisničko ime", "Lozinka", and "Potvrda lozinke", all with masked characters. Below the fields, a note states: "Korisničko ime mora sadržati najmanje 3 karaktera. Korisničko ime mora da počne slovom, a završi slovom ili brojem." A "NAZAD" button and a green "NASTAVI" button are at the bottom.
- Right Screen (Dodeljen prostor):** The title is "Dodeljen prostor" with a storage icon. The progress bar shows the third of three steps completed. It features a circular progress indicator showing "50 MB" out of "(275 MB)". Below this, a note says: "Podelite prostor za ovaj email nalog. Prostor mozete podeliti i kasnije." A "NAZAD" button and a green "ZAVRSI" button are at the bottom.

# Interface design

**Show a numeric indicator on the progress components**



# Interface design

## Spinners



For **short** processes  
less than 4 seconds

## Progress Bars



For **long** processes  
more than 4 seconds

## When to Display a Spinner

If process is...

< 1 seconds

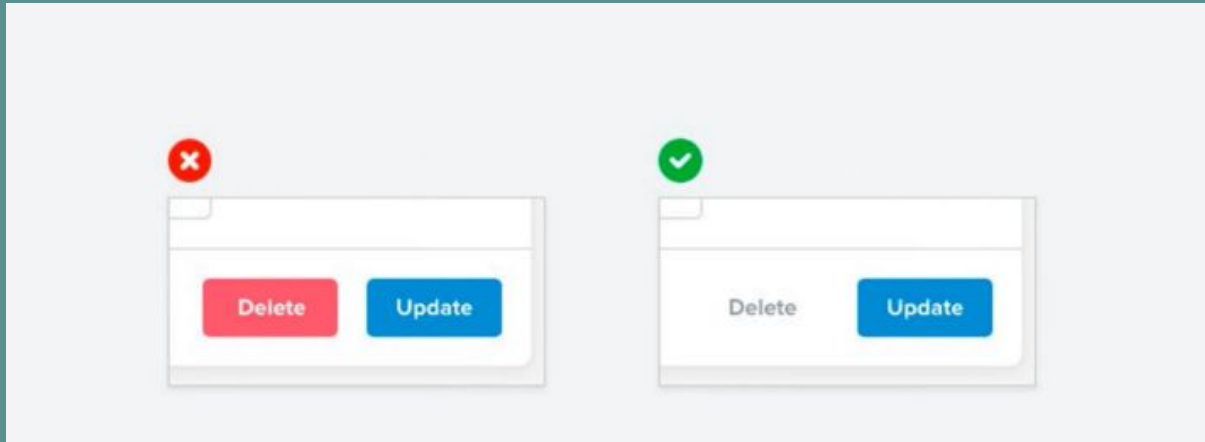
NONE

1 - 4 seconds

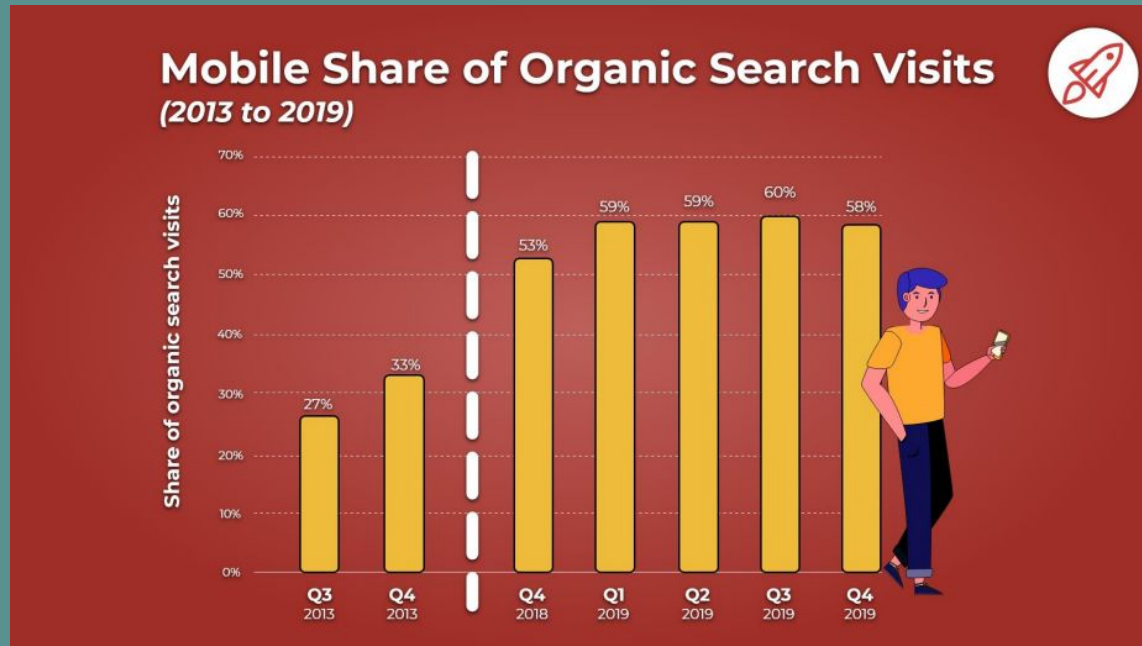


# Interface design

- Use color as a key element for Call to Action



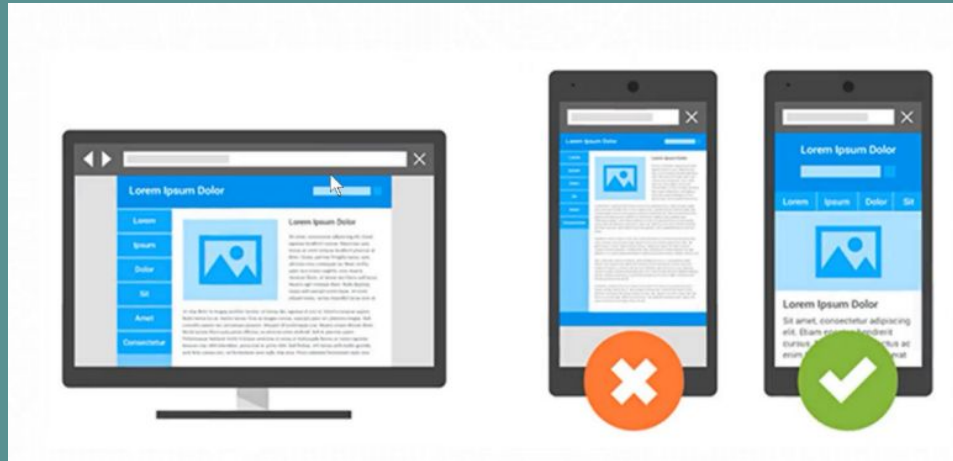
# Interface design





# Interface design

## Mobile share of organic search engine visits in the United States





# Interface design

## UX

- <https://uxplanet.org/best-practices-in-ux-design-for-a-website5159a84dfd7d>
- <https://www.intechnic.com/blog/100-ux-design-pro-tips-from-userexperience-master/>



# Interface design

## Design

- <https://www.awwwards.com/websites/ui-design/>
- <https://duallstudio.com/>

## Image banks

- <https://stocksnap.io/>
- <https://www.pexels.com/pt-br/>
- <https://unsplash.com/>



# Interface design

## Fonts

- <https://fonts.google.com/>
- <https://www.fontsquirrel.com/>
- <https://www.fontspace.com/>

## Icons

- <https://fontawesome.com/>
- <https://iconmonstr.com/>
- <https://www.flaticon.com/>



# Interface design

## Colors

- <https://visme.co/blog/website-color-schemes/>
- <https://colors.co/>
- <https://colorhunt.co/>

## Frameworks

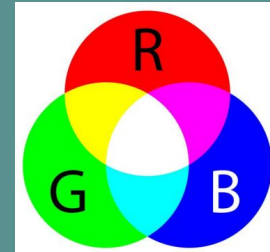
- <https://getbootstrap.com/>
- <https://materializecss.com/>
- <https://fezvrasta.github.io/bootstrap-material-design/>
- <https://material-ui.com/pt/#/>

# Interface design

## Color

- RGB / Hexadecimal

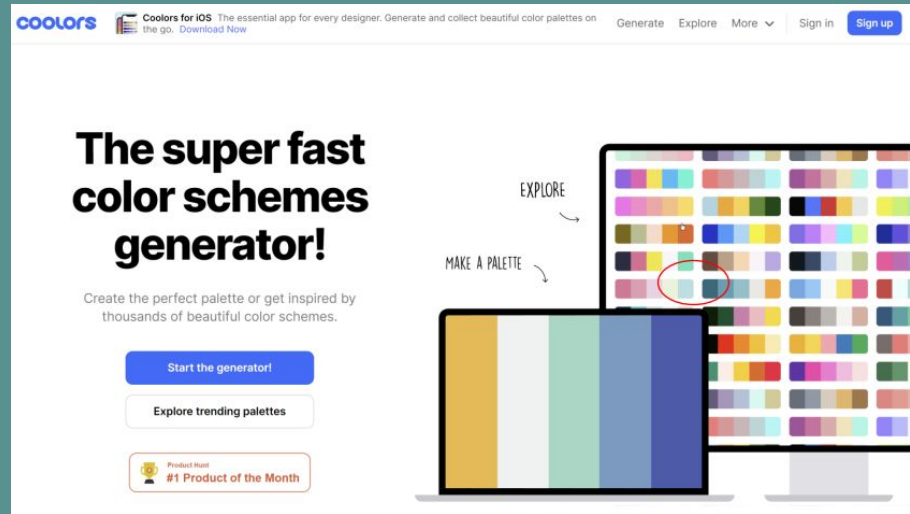
#FF0000	RGB (255, 0, 0)
#FF7F00	RGB (255, 127, 0)
#FFFF00	RGB (255, 255, 0)
#00FF00	RGB (0, 255, 0)
#0000FF	RGB (0, 0, 255)
#4B0082	RGB (75, 0, 130)
#8F00FF	RGB (143, 0, 255)



# Interface design

## Color Paletts

- <https://colors.co/>





# Interface design

## Fonts

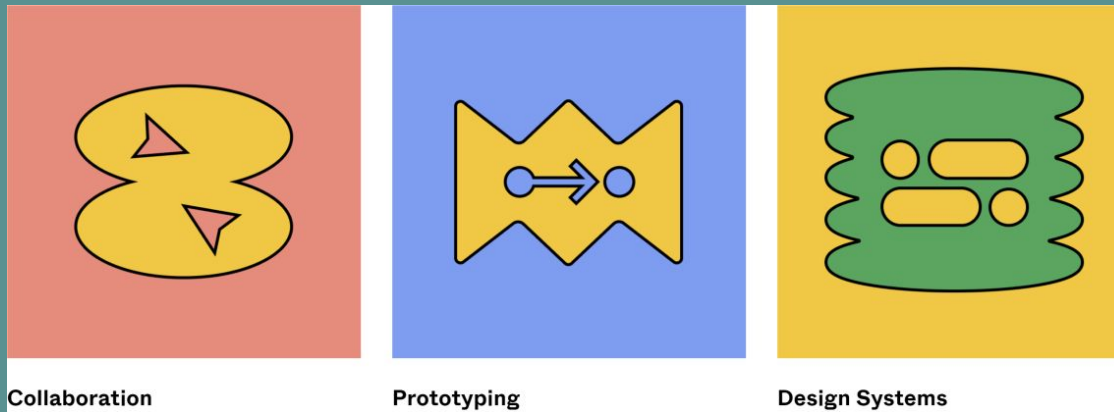
- Fonts are essential elements in design
- Use standard fonts (installed on most computers)
- Or use Google fonts (css fonts)
- <https://fonts.google.com/>





# Interface design

## Figma





# Interface design

## Figma

Register for a free Educational account

- <https://www.figma.com/education/apply/>



# Interface design

## Figma

### Course on Coursera (33 hours)

- Register as listener (you don't need to pay)
- Week 1: Starting to create mockups
- Week 2: Applying visual design principles to mockups



# Today's plan

**Activity n°1**

**Reproduce/Redesign HBO APP**



# Themes for Projects IPB+

## Education:

- Gamification and learning games (eg. kahoot.com)
- Collaborative whiteboard (eg. jamboard.google.com)
- Polls and Word cloud (eg. mentimeter.com, answergarden.ch)
- Continuous assessment registration



# Themes for Projects IPB+

## Administration

- Activities management and spaces reservation
- Resources reservation
- Double degree management



# Themes for Projects IPB+

## Research

- Projects registration and management
- Researchers community network




# Themes for Projects IPB+

## Campus life

- IPB+ community (Communication platform – eg. Whatsapp, Telegram style PWA application)
- Alumni network
- Mentoring program





# Sign In

Email Address

Password

👁

SIGN IN

SIGN IN WITH A PROVIDER

Don't Have an Account? Sign Up

Forgot Password? Need help signing in?

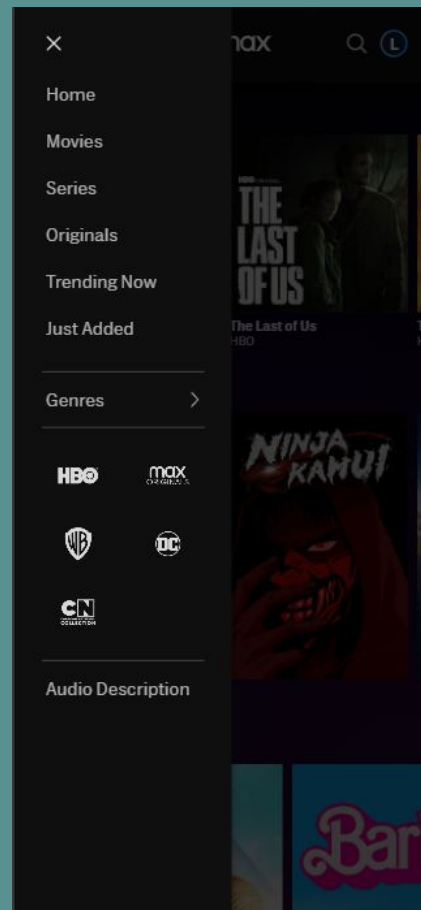
Privacy Policy

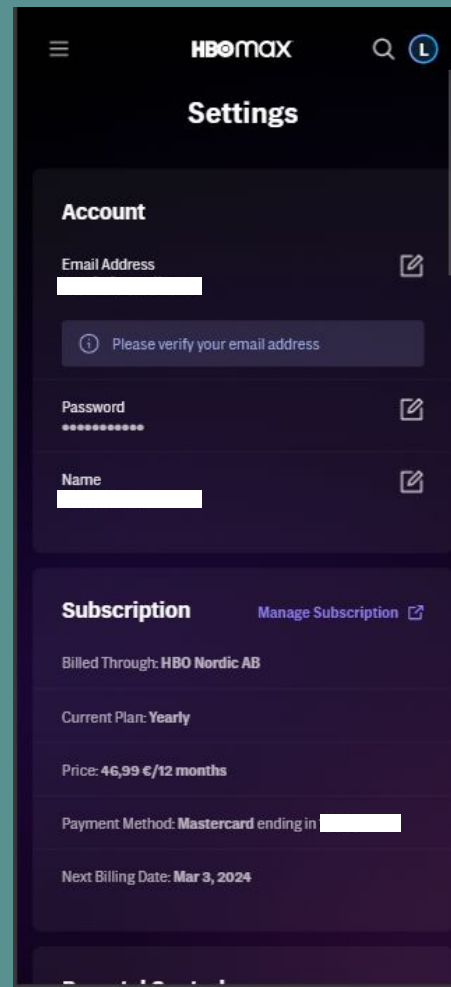
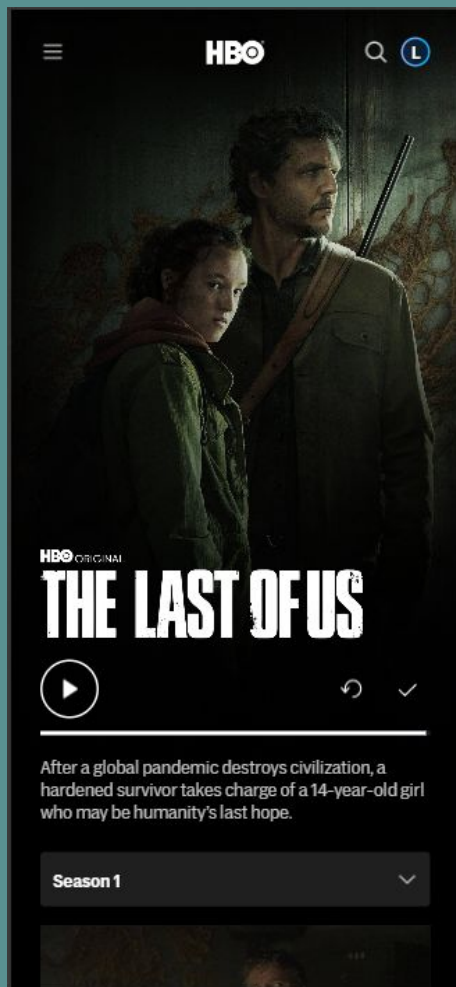
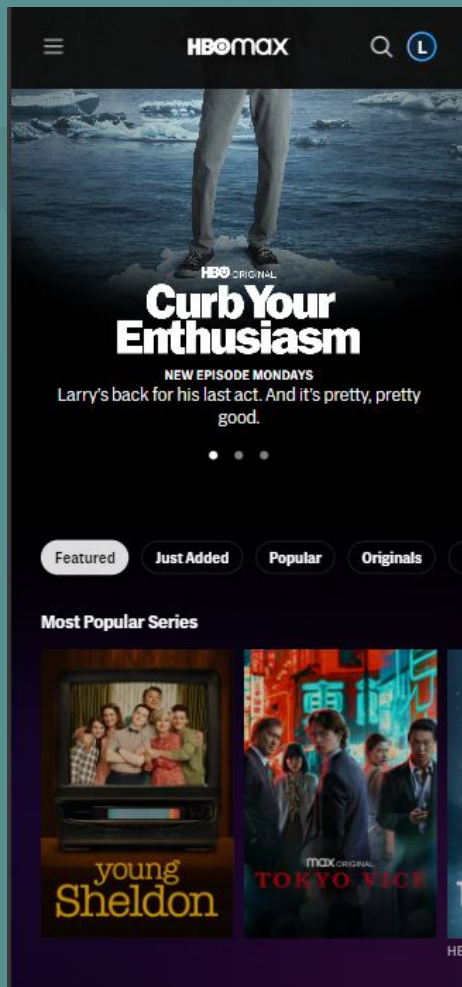
Terms of Use

Info

English (US)

▼







## Week 3

### Constitution of working groups

- 4 to 5 elements
- Fill the file (**Google Drive**)
- Meeting with the groups



## Week 4

### Assignment of themes

- Meeting with the groups
- User stories and first sketches of the app (must be delivered until 21:00 pm by the team leader at dropbox Virtual)
- First mockups



## Week 5

### Introduction to Android and Kotlin

- Please install Android Studio at home



# Mobile applications

## Introduction to Android and Kotlin

- MOOC Android basics Kotlin (18/03 to 08/04):

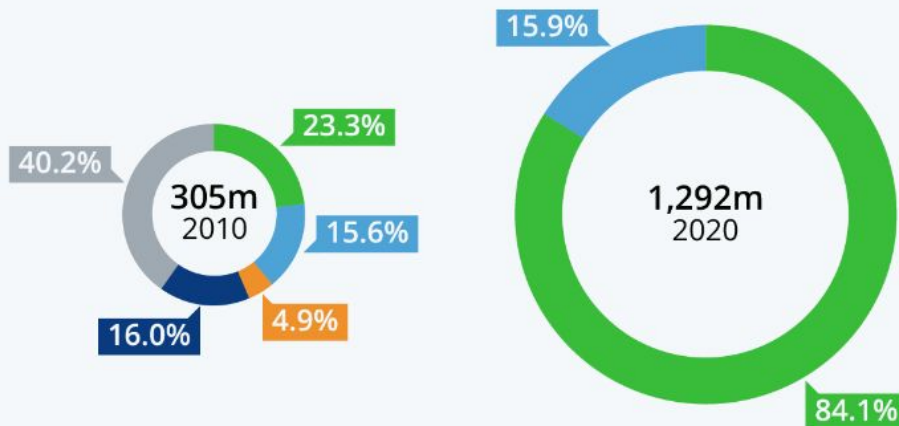
<https://developer.android.com/courses/android-basics-compose/course>

# Mobile applications

## The Smartphone Duopoly

Worldwide smartphone market share by operating system (based on unit shipments)

● Android ● iOS ● Windows Phone ● BlackBerry ● Others

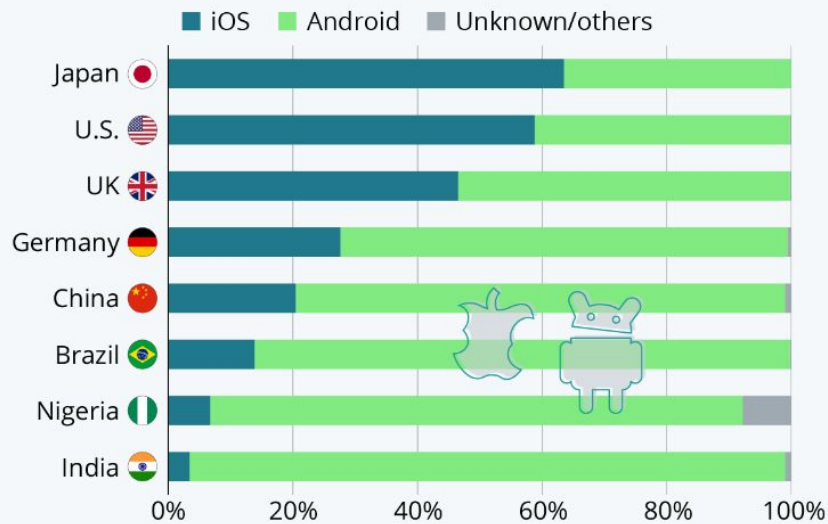


Source: [statista](https://www.statista.com)

# Mobile applications

## Apple or Android Nation?

Mobile operating systems market share in selected countries (as of July 2020)



Source: [statista](#)



# Mobile applications



Source: [statista](https://www.statista.com)

# Mobile applications

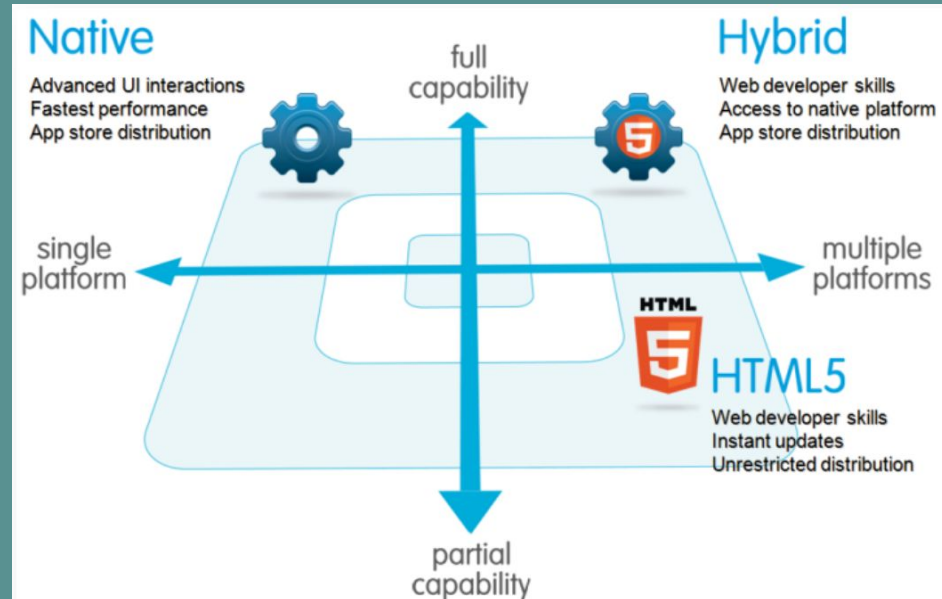
## Mobile applications development

1. Native
2. HTML5
3. Hybrid



# Mobile applications

## Mobile applications development





# Mobile applications

## Native applications:

- Developed using an SDK
- **Distribution** - Easy distribution on the App Store
- **Sell** - can be free or paid
- **Functionality** - Allows to use of all hardware and UI features
- **Connectivity** - not require connectivity to Internet
- **Multi-platform** - only are developed for one platform and not allow portability



# Mobile applications

## Applications HTML5

- **Developed** with Web technologies (HTML5, CSS and Javascript)
- **Distribution** - Providing a link. Can't be published on App Store
- **Sell** - not available on the App Store
- **Functionality** - do not allow the use of hardware resources and UI
- **Connectivity** - Need a permanent Internet connection
- **Multi-platform** - Work on any platform that supports HTML5



# Mobile applications

## Hybrid applications

- **Developed** with Web technologies (HTML5, CSS and Javascript)
- **Compile** for each platform using a toolkit
- **Distribution** - can be distributed in the App Store
- **Sell** - can be free or paid

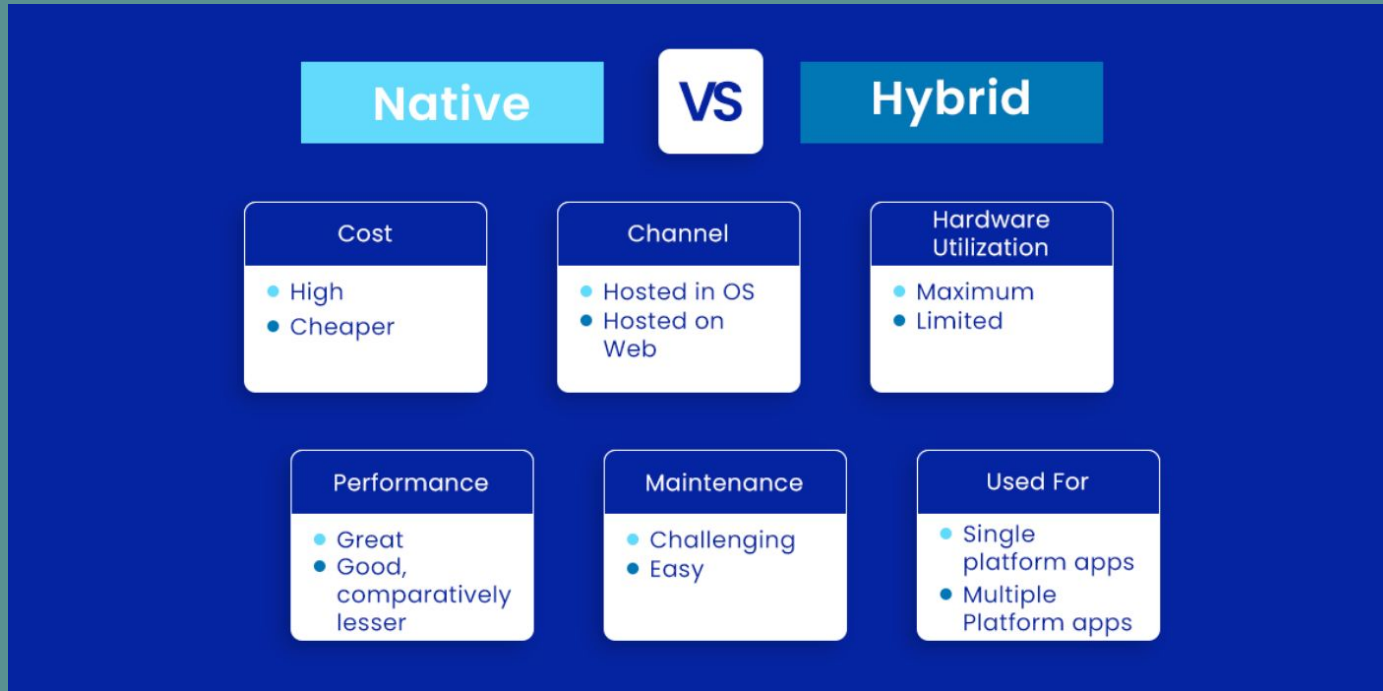


# Mobile applications

## Hybrid applications

- **Functionality** - Allows you to use some hardware features and UI depending on the toolkit
- **Connectivity** - Is not require a permanent connectivity to Internet
- **Multi-platform** - can be compiled for various platforms

# Mobile applications







# Mobile applications

## Cross Platform Native Development

- React Native
- Flutter
- Ionic

More details: [Most used frameworks](#)



# Mobile applications

## Android Development

- Android is an open source platform developed by Google in 2007
- Maintained by the OHA (Open Handset Alliance), a group of more than 40 companies
- Based on Java language and XML
- Alternative language: Kotlin, since 2017



# Mobile applications

## Android Development

- Android is an open source platform developed by Google in 2007
- Maintained by the OHA (Open Handset Alliance), a group of more than 40 companies
- Based on Java language and XML
- Alternative language: Kotlin, since 2017

# Mobile applications



WEAR

## Android Wear

Small, powerful devices, worn on the body.  
Useful information when you need it most.  
Intelligent answers to spoken questions. ....



TV

## About Android TV

Bring your apps, games, and content to the  
biggest screen in the house.



AUTO

## Android Auto

Let drivers listen to and control content in  
your music and other audio apps. Allow  
drivers to hear and respond to your ....



# Mobile applications

## General structure of the Android platform:

- Application framework - to provide fulfillment and replacement components
- Dalvik virtual machine - optimized for mobile devices
- Integrated browser based on webkit engine



# Mobile applications

## General structure of the Android platform:

- Based graphics 2D and 3D libraries, OpenGL ES specification
- SQLite to store structured data
- Multimedia support for audio, video and image formats (MPEG4,
- H.264, MP3, AAC, AMR, JPG, PNG, GIF)

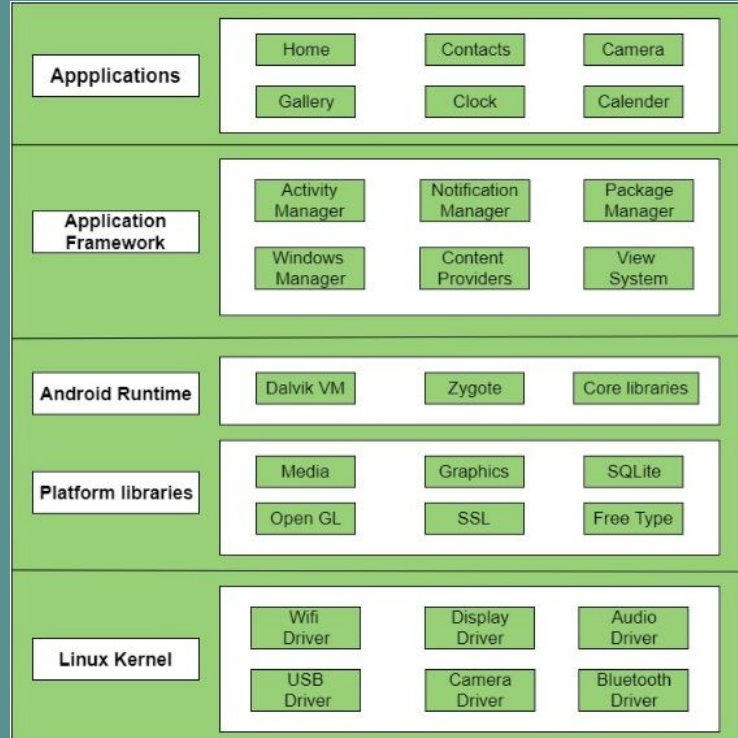


# Mobile applications

## General structure of the Android platform:

- GSM phone (hardware dependent)
- Bluetooth, EDGE, 3G, 4G and WiFi (hardware dependent)
- Camera, GPS, compass, gyroscope, accelerometer (hardware dependent)
- dependent)

# Mobile applications







# Mobile applications

Introduction to **Kotlin**:

- Let's use the Kotlin playground to do some exercises:
  - <https://play.kotlinlang.org/>



# Mobile applications

Introduction to **Kotlin**:

1. Create array of numbers and use the map function to multiply the values by 2
2. Use the filter function to get numbers greater than 10
3. Create a function that takes an argument, and prints it to the console this value



# Mobile applications

Introduction to **Kotlin**:

4. Create object to store information about a Student: name, age, height, course, school, classification
5. Create array of objects, based on previous number
6. Function that prints the name of students from a given school as an argument on the console
7. Function that indicates how many students passed each school



# Mobile applications

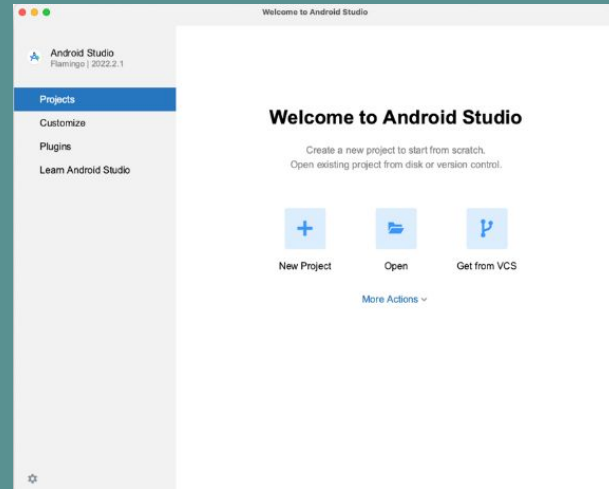
Introduction to **Kotlin**:

8. Save the code to a file and submit it on dropbox ([virtual.ipb](https://www.dropbox.com/s/00000000000000000000000000000000/virtual.ipb)).

# Mobile applications

## Setup Android Studio

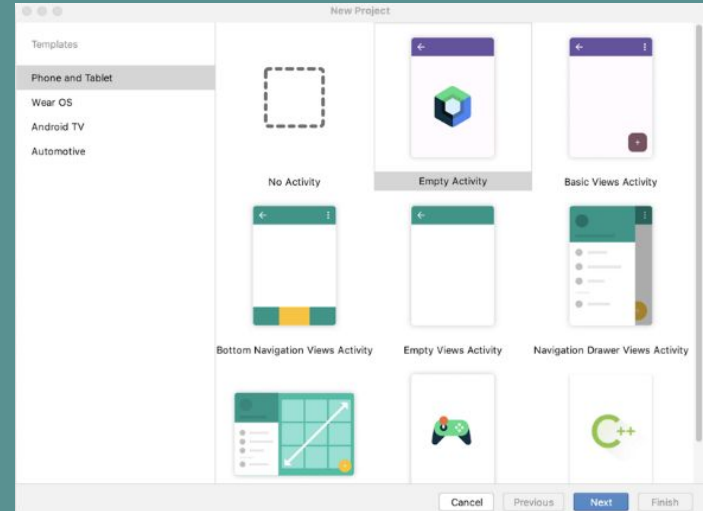
1. Click New Project.



# Mobile applications

## Setup Android Studio

1. Select Empty Activity template
2. Give a name to project.
3. Choose API 24 (Minimum SDK)





# Mobile applications

## Understanding the code

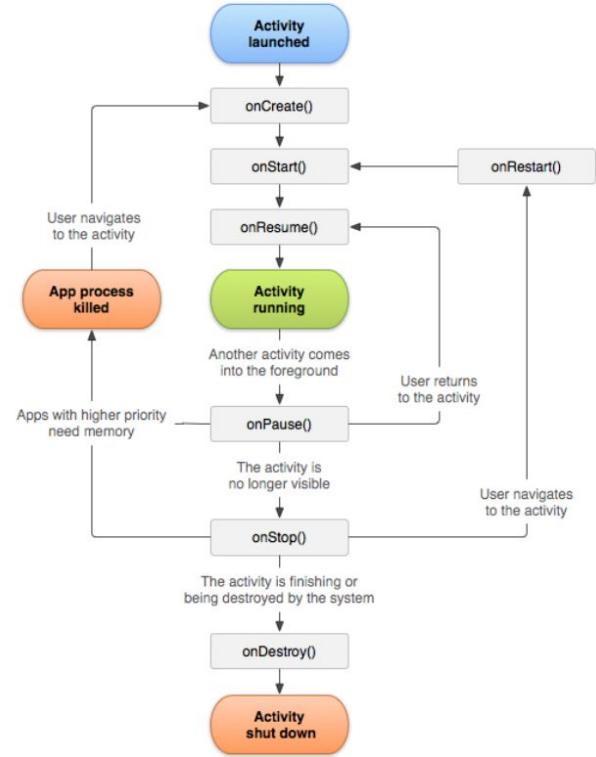
### Whats is an activity?

The **Activity** class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model.

Docs: [Activities](#)

## Activity-lifecycle concepts

## Activity-lifecycle concepts



**Figure 1.** A simplified illustration of the activity lifecycle.





# Mobile applications

## Understanding the code

### ***OnCreate()***

Called when the activity is first created. This is where you should do all of your normal static set up: create views (ui), bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one.

Always followed by **onStart()**.



# Mobile applications

## Understanding the code

### *setContent()*

The **setContent()** function within the **onCreate()** function is used to define your layout through composable functions. All functions marked with the **@Composable** annotation can be called from the **setContent()** function or from other Composable functions. The annotation tells the Kotlin compiler that this function is used by **Jetpack Compose** to generate the UI.



# Mobile applications

## What is Jetpack Compose?

Jetpack Compose is Android's recommended modern toolkit for building native UI. It simplifies and accelerates UI development on Android. Quickly bring your app to life with less code, powerful tools, and intuitive Kotlin APIs.



# Mobile applications

## What is Jetpack Compose?

- Released: July of 2021
- Highly adopted during 2023
- Standard for new apps
- Succeeds XML views



# Mobile applications

## XML views vs Jetpack compose

Aspect	Jetpack Compose	XML Architecture
Approach to UI design and structure	Contemporary, declarative, and composable	Imperative and hierarchical
UI Description	Defined using Kotlin functions	Specified in XML files
Reactivity	Reactive approach based on app's state	Requires manual management of view state changes and updates
Reusability	Composable UI elements as reusable blocks	View hierarchies for layout arrangement
UI Interactions and Animations	Easier and more convenient management through composable functions	Manual handling, more complex and verbose code
Modularity and Maintainability	Encapsulation of behaviors in custom composable functions	Code may become tougher to maintain as the app grows



# Mobile applications

## XML views vs Jetpack compose

Aspect	Jetpack Compose	XML Architecture
Testability	Allows for cleaner and more testable code	May require more effort in testing
Learning Curve	May have a sharper learning curve for developers new to declarative UI	Familiar to developers who are familiar with XML and imperative approach
Performance	Efficient and potentially better performance due to UI optimization techniques	May have performance overhead in large and complex layouts
Development Community	Growing community and active development support	Mature and well-established community with extensive resources
IDE Support	Good IDE support with Android Studio	Fully integrated support with Android Studio
Backward Compatibility	Potential challenges in maintaining backward compatibility	Relatively better backward compatibility with older devices



# Mobile applications

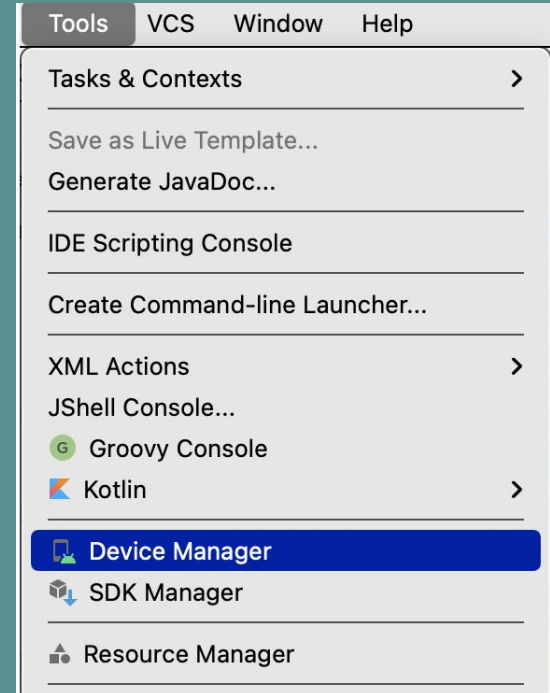
## Jetpack compose Components

- Find the list of components
  - <https://developer.android.com/jetpack/compose/components>

# Mobile applications

## Let's run our app

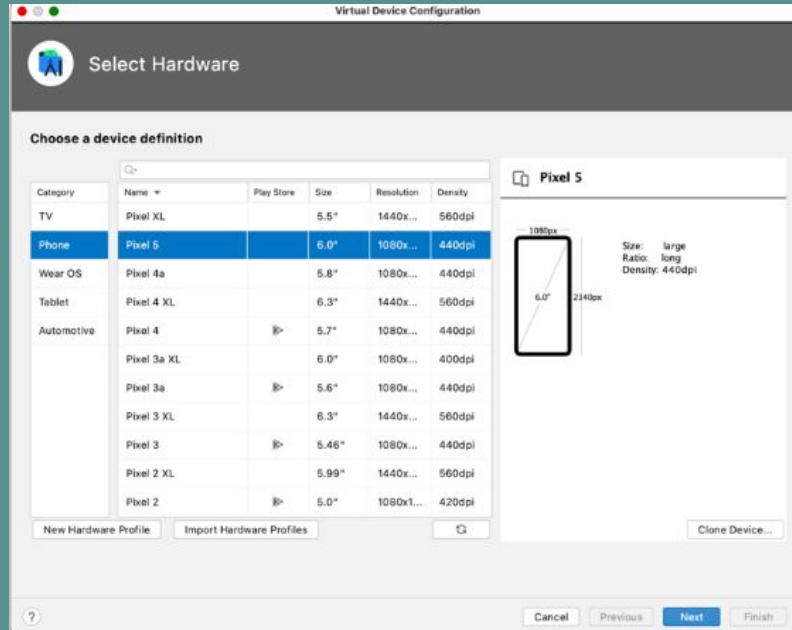
- Run your app on the Android Emulator
- Create an AVD
- Give it a try





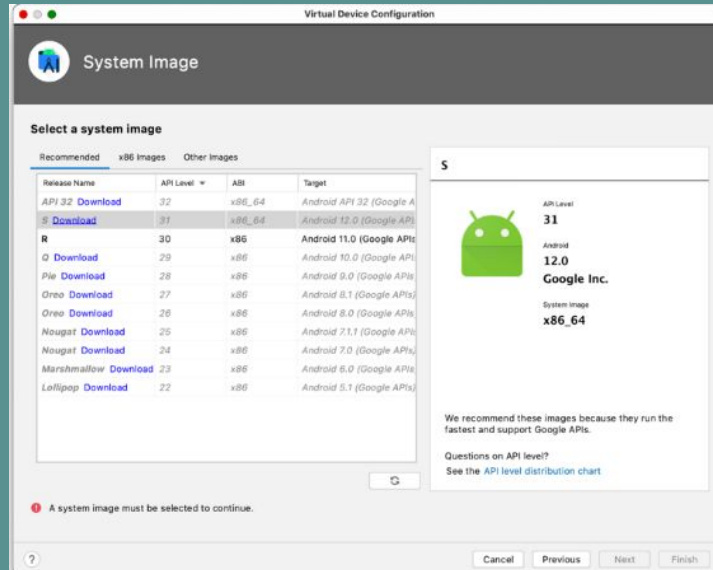
# Mobile applications

Let's run our app



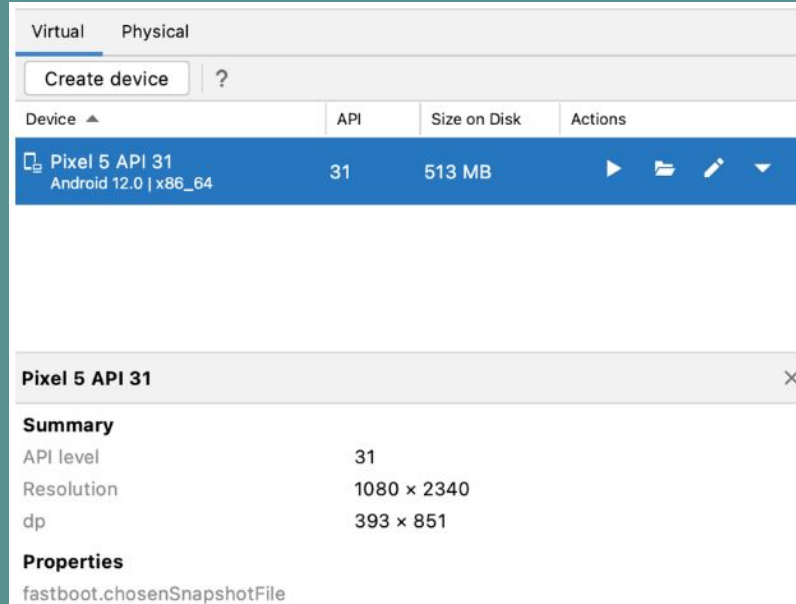
# Mobile applications

Let's run our app








# Mobile applications

## Let's run our app



The screenshot shows the Android Studio interface with the 'Virtual' tab selected. At the top, there are tabs for 'Virtual' and 'Physical'. Below them is a 'Create device' button and a help icon. A table lists the available virtual devices:

Device	API	Size on Disk	Actions
 Pixel 5 API 31 Android 12.0   x86_64	31	513 MB	   

Below the table, a detailed view for 'Pixel 5 API 31' is shown, including a close button (X). The view is divided into 'Summary' and 'Properties' sections.

**Summary**

API level	31
Resolution	1080 x 2340
dp	393 x 851

**Properties**

fastboot.chosenSnapshotFile



# Mobile applications

## Managing State and Jetpack Compose

- **What is a state?**

State in an app is any value that can change over time. This is a very broad definition and encompasses everything from a Room database to a variable in a class.



# Mobile applications

## Managing State and Jetpack Compose

- **State and composition**

Compose is declarative and as such the only way to update it is by calling the same composable with new arguments. These arguments are representations of the UI state. Any time a state is updated a recomposition takes place. As a result, things like `TextField` don't automatically update like they do in imperative XML based views. A composable has to explicitly be told the new state in order for it to update accordingly.



# Mobile applications

## Managing State and Jetpack Compose

- **State in composables**

Composable functions can use the ***remember*** API to store an object in memory. A value computed by ***remember*** is stored in the Composition during initial composition, and the stored value is returned during recomposition. ***remember*** can be used to store both mutable and immutable objects.



# Mobile applications

## Managing State and Jetpack Compose

Any changes to value schedules recomposition of any composable functions that read value. There are three ways to declare a ***MutableState*** object in a composable:

- ***val mutableState = remember { mutableStateOf(default) }***
- ***var value by remember { mutableStateOf(default) }***
- ***val (value, setValue) = remember { mutableStateOf(default) }***

These declarations are equivalent, and are provided as syntax sugar for different uses of state. You should pick the one that produces the easiest-to-read code in the composable you're writing.



# Mobile applications

## What we learned?

- Create a basic UI
- TextInput, Button
- Basic State Management





# Mobile applications

## Advanced UI Components - Lists and grids

### Lists and grids

- **What is a Lazy Column?**

A **LazyColumn** is a part of Jetpack Compose, Android's modern UI toolkit.



# Mobile applications

## LazyColumn

Imagine you have a long list of items that you want to display on the screen, like a list of contacts, messages, or products. If you try to load and display all items at once, it might slow down your app because it's a lot of information to process and show.



# Mobile applications

## LazyColumn

Here's where ***LazyColumn*** comes to the rescue! It's smart and "lazy." It only loads and displays the items that fit on the screen.

As the user scrolls down, ***LazyColumn*** cleverly loads more items on-the-go. This way, your app remains smooth and responsive, providing a better user experience.



# Mobile applications

## AlertDialog

An ***AlertDialog*** in Android's Jetpack Compose is a type of pop-up window that appears in front of the main content of the app. It's used to capture user attention and convey important information or get user input.



# Mobile applications

## **AlertDialog**

For example, you might use an ***AlertDialog*** to confirm user actions, show error messages, or ask for user choices or decisions.



# Mobile applications

## Understanding Lambdas - Kotlin

Lambdas are a feature in Kotlin that allows you to treat functionality as a method argument, or create concise ways to express functions/methods.



# Mobile applications

## Understanding Lambdas - Kotlin

In simpler terms, a lambda is like a mini-function that you can create on the fly without needing a name.

Lambdas are useful for short, simple operations that are used right away, often as arguments to other functions.



# Mobile applications

## Understanding Lambdas - Kotlin

Syntax:

A lambda expression is always enclosed in curly braces {}. It may have parameters, and it may return a value. The parameters are defined at the start of the lambda, followed by an arrow ->, and then comes the body of the lambda.





# Mobile applications

## Understanding Lambdas - Kotlin

The program below has a lambda expression that accepts two integers as parameters, and returns the product of those two integers.

```
fun main(args: Array<String>) {  
    val product = { a: Int, b: Int -> a * b }  
    val result = product(9, 3)  
    println(result)  
}
```

# Mobile applications

## Understanding Lambdas - Kotlin

No params and no return.

```
val printMessage: () -> Unit = {  
    println("Hello, world!")  
}  
  
fun main() {  
    printMessage() // Output: Hello, world!  
}
```



# Mobile applications

## Data Classes - Kotlin

Data classes in Kotlin are primarily used to hold data. For each data class, the compiler automatically generates additional member functions that allow you to print an instance to readable output, compare instances, copy instances, and more. Data classes are marked with ***data:***



# Mobile applications

## Data Classes - Kotlin

```
data class User(val name: String, val age: Int)
```

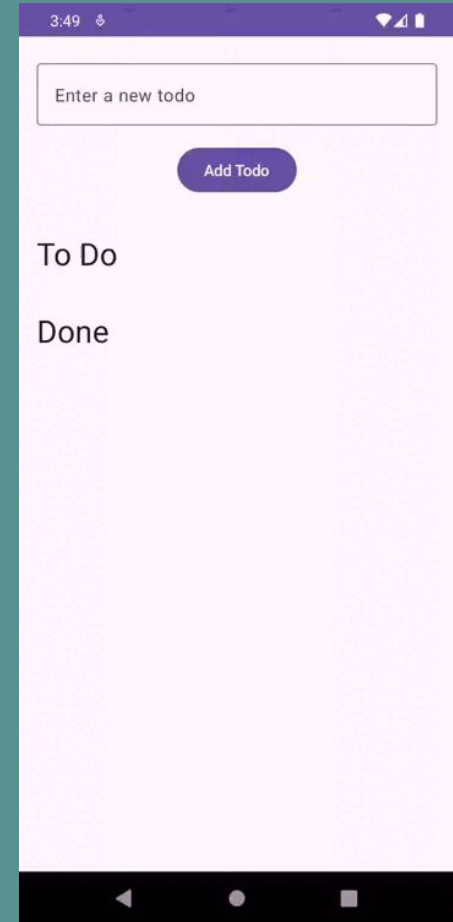
# Mobile applications

## Individual Activity n°2

### Create a TodoList.

1. Use data class for ***todo***
2. Confirmation prompt required before deleting a to-do item.
3. Make the app look better

[Todo app video](#)





# Mobile applications

## How to organize your app?

### Understanding *MVVM*

MVVM stands for Model-View-ViewModel. It is an architectural pattern used in software development, which helps in organizing your code in a way that is easier to understand, test, and maintain.

Let's break down what each component in MVVM does...



# Mobile applications

## 1. Model

- **What it is:** The Model represents the data and business logic of the application. It is responsible for retrieving and storing data, as well as performing any necessary data processing.
- **Example:** If you have an app that displays a list of books, the Model would be responsible for fetching the book data from a database or an online source.



# Mobile applications

## 2. View

- **What it is:** The View is the user interface (UI) of the application. It displays the data to the user and interacts with the user.
- **Example:** In the same book app, the View would be the actual screen that displays the list of books to the user.





# Mobile applications

## 3. ViewModel

- **What it is:** The ViewModel acts as a bridge between the Model and the View. It takes data from the Model, applies UI logic, and then formats it for display in the View.
- **Example:** For the book app, the ViewModel would take the raw book data, and format it nicely for display in the View.



# Mobile applications

## Syntax and Code Examples: MVVM

Here's a simplified example in Kotlin to give you a basic idea of how MVVM works:

### 1. Model

```
data class Book(val title: String, val author: String)
```



# Mobile applications

## 2. ViewModel

```
class BookViewModel {  
    fun getBooks(): List<Book> {  
        // Here you would normally fetch data from a database or online source  
        return listOf(Book("Title1", "Author1"), Book("Title2", "Author2"))  
    }  
}
```

# Mobile applications

## 3. View

Here you need to design your UI, in the case, listing the books.

```
class BookActivity : AppCompatActivity() {  
    private lateinit var viewModel: BookViewModel  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_book)  
  
        viewModel = BookViewModel()  
  
        val books = viewModel.getBooks()  
        // Now you can display the books in your UI  
    }  
}
```



# Mobile applications

## What is a ViewModel Class?

A ViewModel Class in Android is a part of the MVVM architecture, as previously mentioned. It acts as a manager that handles the communication between the app's data and the UI. The ViewModel is responsible for holding and processing all the data needed for the UI while respecting the lifecycle of the app's activities or fragments.



# Mobile applications

## Why is ViewModel Important?

1. **Lifecycle Awareness:** ViewModel is designed to store and manage UI-related data in a lifecycle-conscious way. It allows data to survive configuration changes such as screen rotations.
2. **Separation of Concerns:** ViewModel helps to keep the UI code simple and focused on presenting data, as it takes care of the data handling.



# Mobile applications

## How Does ViewModel Work?

- **Initialization:** A ViewModel is usually initialized in an activity or fragment. It survives configuration changes and is destroyed when the activity or fragment is permanently removed.
- **Data Handling:** ViewModel retrieves data from the Model, holds it, and the View observes this data to update the UI accordingly.



## Mobile applications

### **What are Repositories in Android Development?**

In the context of Android development, particularly when following the MVVM (Model-View-ViewModel) architecture, a Repository is a class that acts as a clean API for data access to the rest of the application.

It abstracts the origin of the data, which can come from a network source, caching, or a local database.





# Mobile applications

## MVVM Architecture: Why Use Repositories?

- **Decoupling:** Repositories allow for decoupling of the data sources from the rest of the application. The ViewModel interacts with a Repository, and it doesn't need to know where the data comes from.
- **Data Aggregation:** A Repository can manage and coordinate data from multiple sources, providing a unified API.
- **Offline Capability:** Repositories can cache network data, allowing apps to work offline and providing a better user experience.



# Mobile applications

## Example of repository

Imagine you are building an app that displays user profiles. The data can come from a local database or a network source.

### 1. Defining a Repository

```
class UserRepository(private val userDao: UserDao, private val userService: UserService) {  
    fun getUser(userId: String): LiveData<User> {  
        // Logic to fetch data from network or local database  
    }  
}
```



# Mobile applications

## 2. Using the Repository in a ViewModel

```
class UserViewModel(private val repository: UserRepository) : ViewModel() {  
    fun getUser(userId: String): LiveData<User> {  
        return repository.getUser(userId)  
    }  
}
```



## Mobile applications

### 3. Accessing Data in the View

The View observes the data provided by the ViewModel and updates the UI accordingly.



## Mobile applications

Let's do a super basic example using MVVM.

You can find the example at IPB Virtual



## Mobile applications

Meeting with the teams to check:

- Mockups
- User Stories
- Use cases

Final Version:

Must be submitted until 28th of April by team leader



# Mobile applications

## Gradle Scripts in Android Studio

Gradle scripts are essential files that manage the build process of an Android application.

Here's a brief overview:

1. Project-level build.gradle (Top-level): Located at the root of your project, this file defines configurations common across all modules, such as repositories and classpath dependencies.



# Mobile applications

## Gradle Scripts in Android Studio

2. Module-level build.gradle (App-level): Located in each module, this file specifies module-specific configurations like application ID, SDK versions, and dependencies.
3. Gradle Properties: These are configurations that apply globally across tasks and projects, like memory settings.





# Mobile applications

## Gradle Scripts in Android Studio

4. Gradle Tasks: Tasks are actions that Gradle executes during the build process, such as compiling and packaging the application.

In essence, Gradle scripts configure, customize, and automate the steps in compiling, building, and packaging your Android application in Android Studio.



# Mobile applications

## Dependencies in Android Studio

Dependencies refer to external libraries or modules that your Android project relies on to function correctly.

These dependencies can provide various functionalities, such as network operations, image loading, data persistence, and more, allowing you to build feature-rich applications without reinventing the wheel.



# Mobile applications

## Dependencies in Android Studio

### 1. Library Dependencies

- Kotlin and Android Libraries: Pre-built libraries specifically designed to be used with Kotlin and Android, such as Kotlin Coroutines for asynchronous programming or Retrofit for network operations.
- Example: Adding Retrofit as a dependency in your Android project to handle HTTP requests and responses.



# Mobile applications

## Dependencies in Android Studio

### 2. SDK Dependencies

- Android Support Libraries: Libraries like AndroidX or Jetpack components, which offer backward-compatible versions of new Android features and other helpful utilities.
- Example: Using AndroidX RecyclerView to display lists of items in your application.



# Mobile applications

## Dependencies in Android Studio

### 3. Remote Dependencies

- Maven and JCenter: Dependencies hosted on remote repositories. Gradle handles the downloading and integration of these libraries automatically.
- Example: Adding a remote dependency from Maven Central in your build.gradle file.



# Mobile applications

## How to Add Dependencies in Kotlin Android Projects

In Kotlin Android projects, dependencies are added in the `build.gradle.kts` (Kotlin DSL) or `build.gradle` (Groovy DSL) file of the app module. Here's how you might add a dependency using

Groovy DSL:

```
dependencies {  
    // Kotlin standard library  
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"  
  
    // Retrofit for network operations  
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'  
  
    // AndroidX RecyclerView  
    implementation 'androidx.recyclerview:recyclerview:1.2.1'  
}
```



# Mobile applications

## **Retrofit: Android Development**

Retrofit is a type-safe HTTP client for Android and Java, and it is highly used in Kotlin-based Android applications.

Developed by Square, Retrofit simplifies the process of consuming JSON or XML data which is transformed into Kotlin or Java objects in your application.



# Mobile applications

## Retrofit: Android Development

### 1. Functionality

- HTTP Requests: Retrofit simplifies the process of making HTTP requests to REST APIs, handling various aspects like URL construction, session handling, and error processing.
- Data Conversion: Retrofit can automatically convert the JSON or XML responses from the APIs into Kotlin objects, making the data easy to use in your application.





# Mobile applications

## Retrofit: Android Development

### 2. Annotations

- Retrofit uses annotations to define API endpoints and HTTP methods (GET, POST, PUT, DELETE, etc.), making the code more readable and concise.



# Mobile applications

## Retrofit: Android Development

### 3. Integration with Coroutines

- Retrofit has seamless integration with Kotlin Coroutines, which allows for easy implementation of asynchronous API requests and responses.



# Mobile applications

## Retrofit: Android Development

### 4. Customization

- Retrofit allows for extensive customization, such as adding headers, authentication, or custom converters, to tailor the HTTP client according to the needs of the application.



# Mobile applications

## Coroutines in Android Development

Coroutines are a powerful feature in Kotlin that makes asynchronous programming more **manageable and concise**.

They help in simplifying the code that deals with operations like network calls, database transactions, and other tasks that can be performed asynchronously.



# Mobile applications

## Coroutines in Android Development

- Coroutines facilitate writing asynchronous code in a sequential manner, making it more readable and understandable.
- They allow you to write non-blocking code, meaning that the execution can be paused and resumed, allowing other tasks to run in the meantime.
- This is particularly useful in UI applications like Android, where you don't want to block the main thread to keep the UI responsive.



# Mobile applications

## Syntax and Components: Coroutines

### 1. Suspend Keyword

- Suspend Functions: Functions marked with the suspend keyword are the building blocks of coroutines. They can be paused and resumed, allowing for non-blocking asynchronous execution.
- Usage: Suspend functions can be invoked from other suspend functions or within a coroutine scope.



# Mobile applications

## Syntax and Components: Coroutines

### 2. Coroutine Builders

- **launch**: Starts a new coroutine without blocking the current thread and returns a reference to the coroutine as a Job.
- **async**: Starts a new coroutine and returns a Deferred, which is a non-blocking cancellable future that represents the result.



# Mobile applications

## Syntax and Components: Coroutines

### 3. Coroutine Scopes

Scopes control the lifetime of coroutines. For example, in Android, you might use a scope tied to the application's lifecycle.





## Mobile applications

**Let's tear down a practical example using the previous concepts.**

**You can find the example at [virtual](#).**



# Mobile applications

## Individual Activity n°3

### Based on Meals App:

1. Choose an API to list something of your interest.
2. You can use a public API (<https://github.com/public-apis/public-apis>) or you can create yourself an endpoint using <https://jsonplaceholder.typicode.com/>
3. Make the list looks good
4. Make some type of filter of your list.



# Mobile applications

## Navigating with Compose

<https://developer.android.com/develop/ui/compose/navigation>

1. The Navigation component for Jetpack Compose is a part of Android Jetpack, a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices.



# Mobile applications

## Navigating with Compose

2. In traditional Android development, navigation is typically handled by using fragments and activities. Jetpack Compose does not use fragments or activities for the UI but instead uses composable functions to define the UI.



# Mobile applications

## Navigating with Compose

3. To support navigation in the context of composable functions, the Navigation component for Jetpack Compose offers a simple and consistent way to implement navigation within your Compose applications.



# Mobile applications

## Navigation - Setup

To support Compose, use the following dependency in your app module's build.gradle file:

```
dependencies {  
    val nav_version = "2.7.7"  
  
    implementation("androidx.navigation:navigation-compose:$nav_version")  
}
```



# Mobile applications

## Individual Activity n°4

### Based on Activity n°3

1. Add a screen of detail for item in list
2. Add navigation between the list and detail