

Didact Documentation

Authors

- Stav Rockah - 307900878
- Roman Smirnov - 312914443

Goals

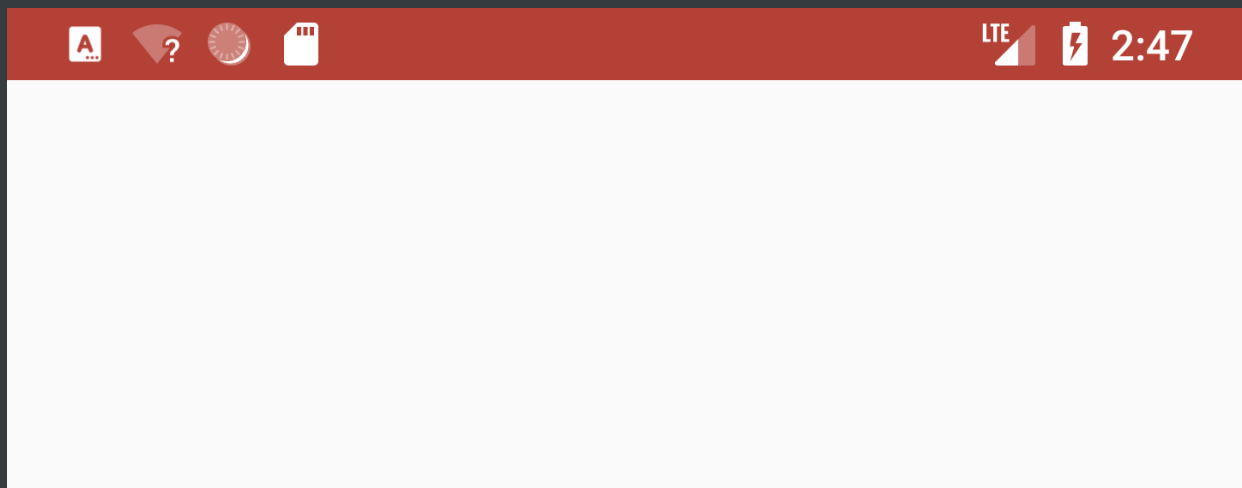
Our goals for the project were to create a server-client app with a reasonably good architecture.

App Description

Our app is an ebook-reader. It allows users to create accounts, and then download and read books. It's composed of an Android client and a Spring server.

App Showcase


Splash Screen





Login Screen


LTE 2:43



Welcome to Didact

Email

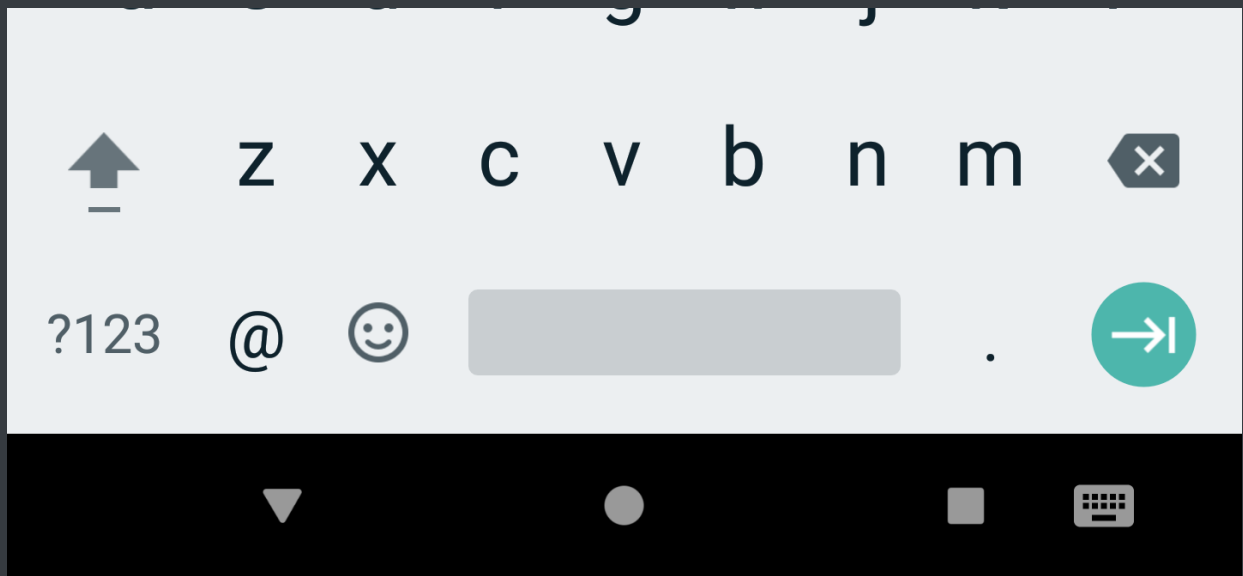
Your password goes here



1234567890

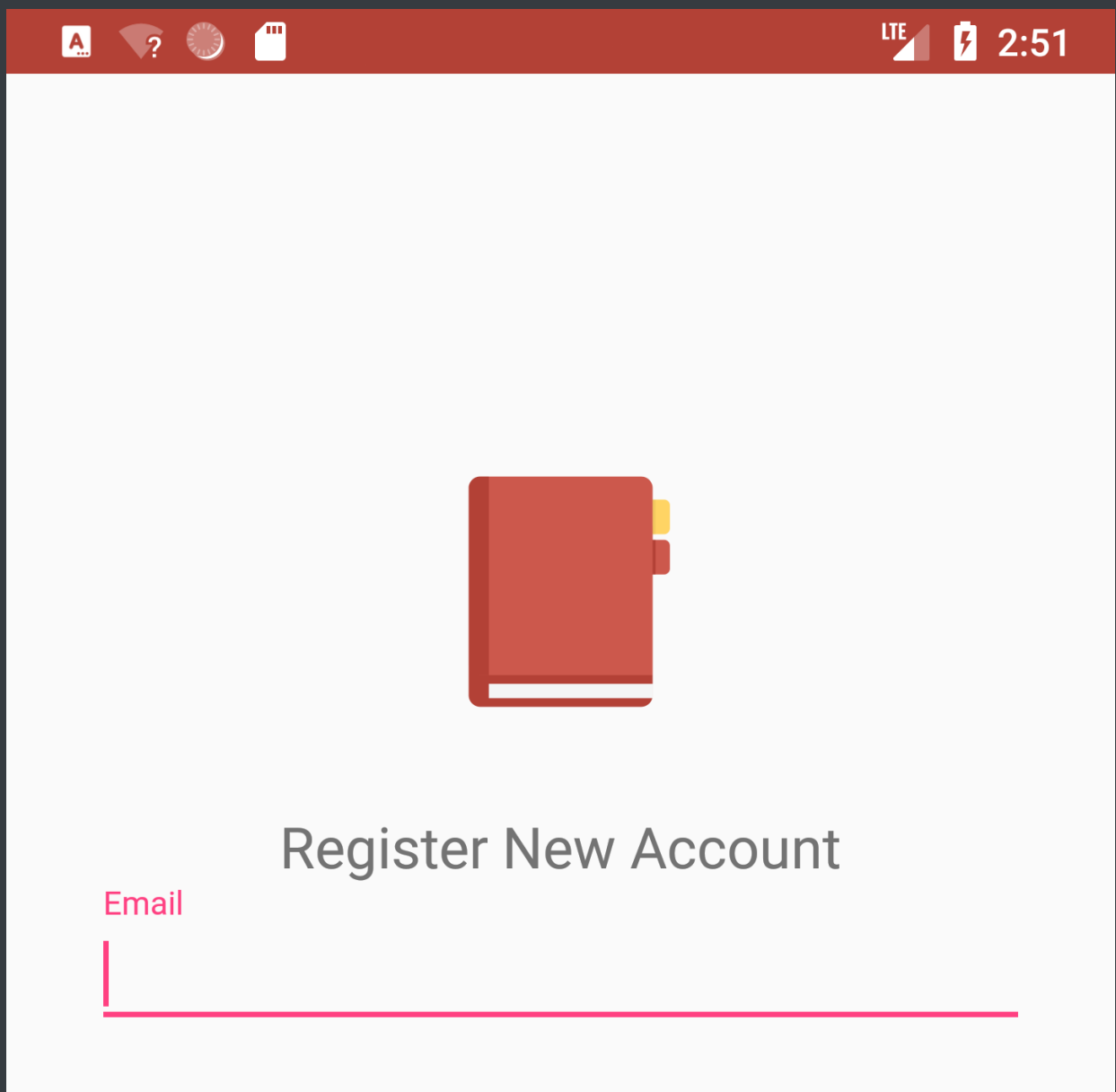
q w e r t y u i o p

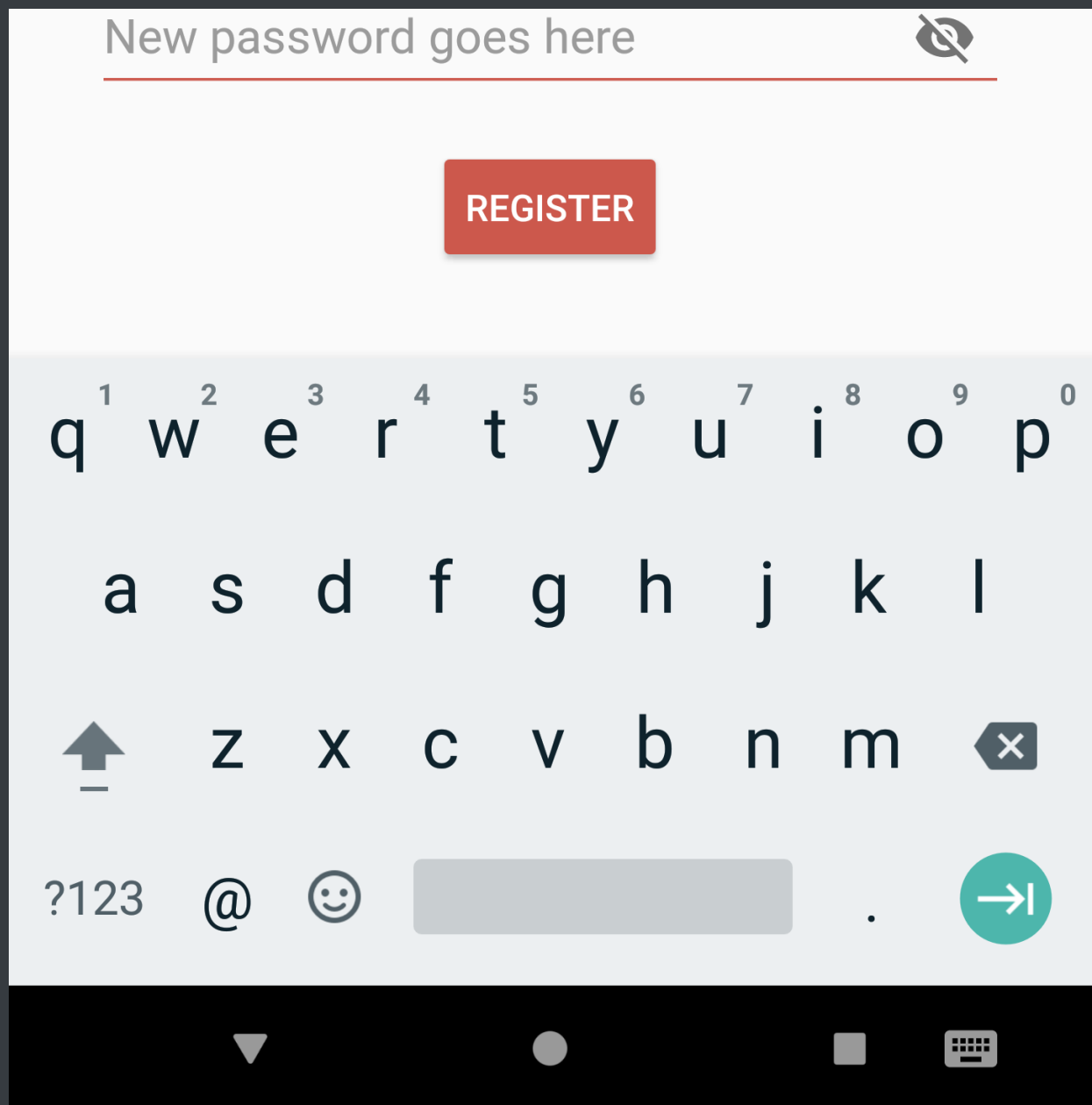
a s d f g h i k l



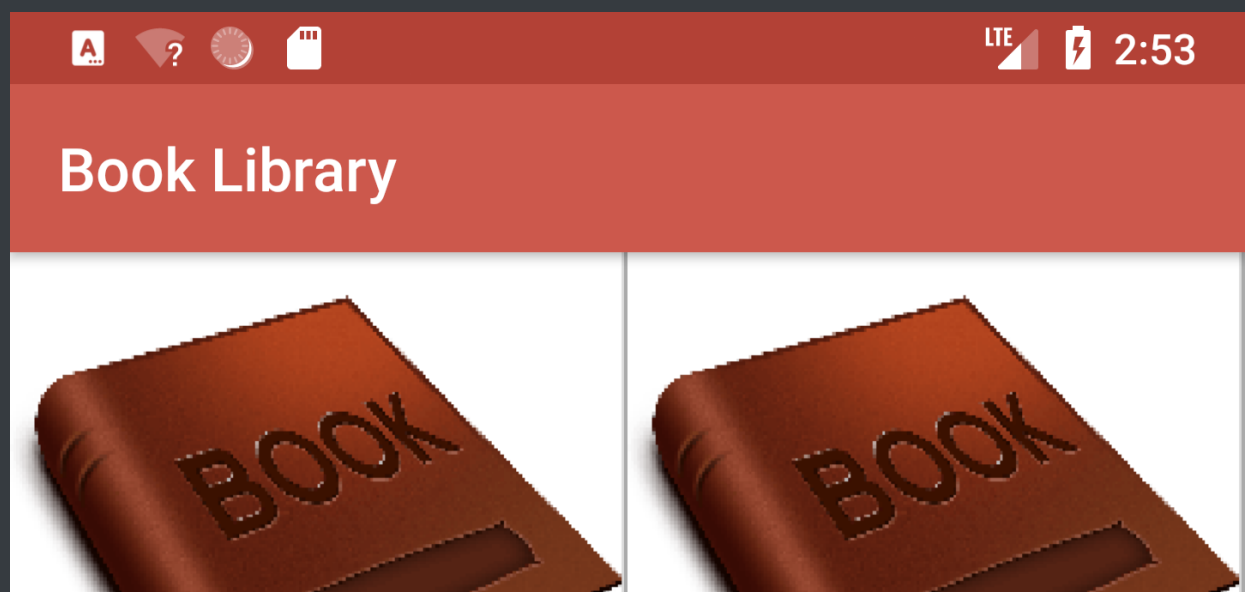
![section_detail)(didact_documentation.assets/section_detail.png)

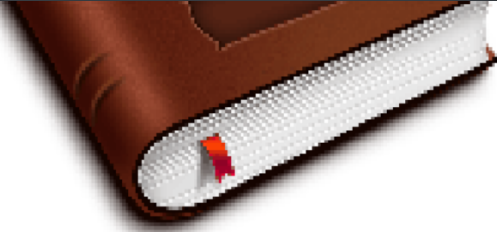
Registration Screen





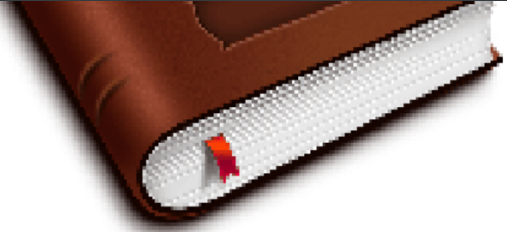
Book Library Screen





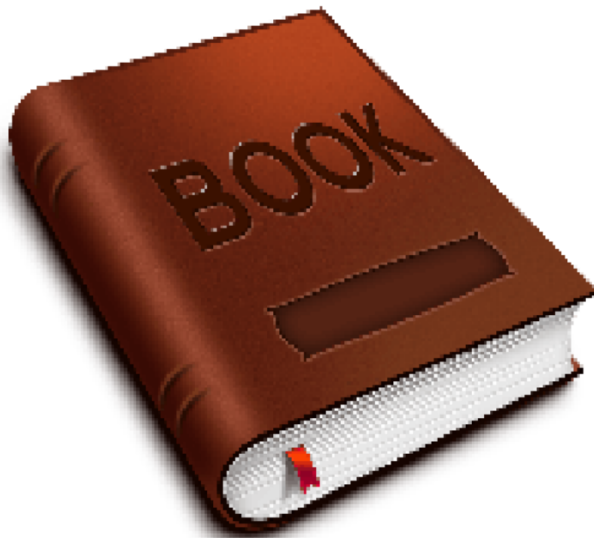
Great Short Works of Herman
Melville

it's about space adventures



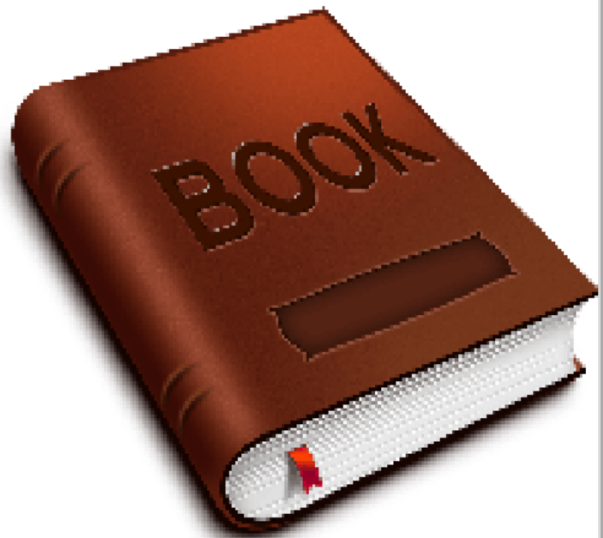
Vance Packard & American
Social Criticism

it's about space adventures



Walt Disney's Uncle Scrooge &
Donald Duck: The Sunken City
(Gladstone Giant Comic Album
Series, #2)

it's about space adventures



Blue Like Jazz: Nonreligious
Thoughts on Christian
Spirituality

it's about space adventures





2:53

Chapters



Cool chapter 1
we going to learn a lot!

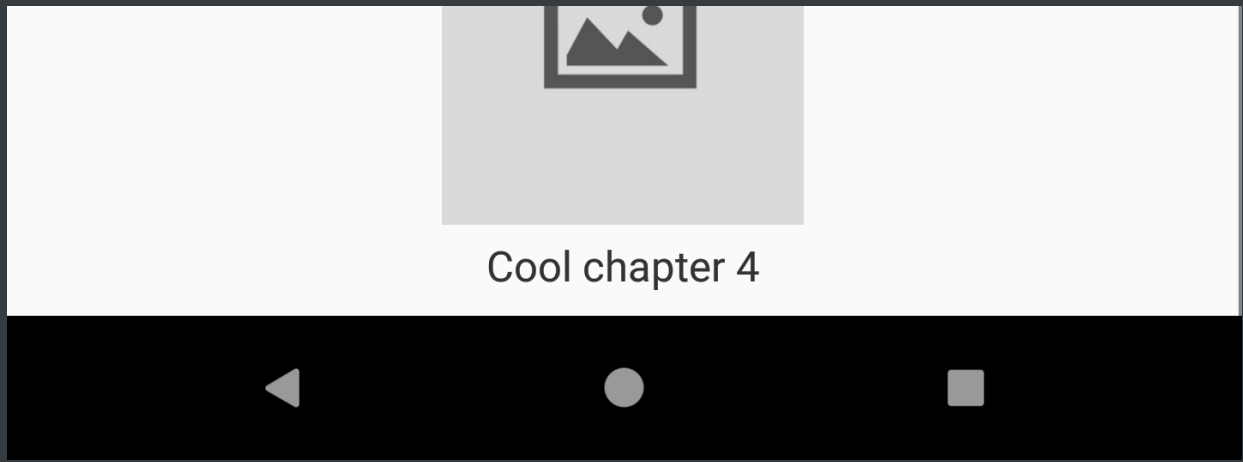


Cool chapter 2
we going to learn a lot!

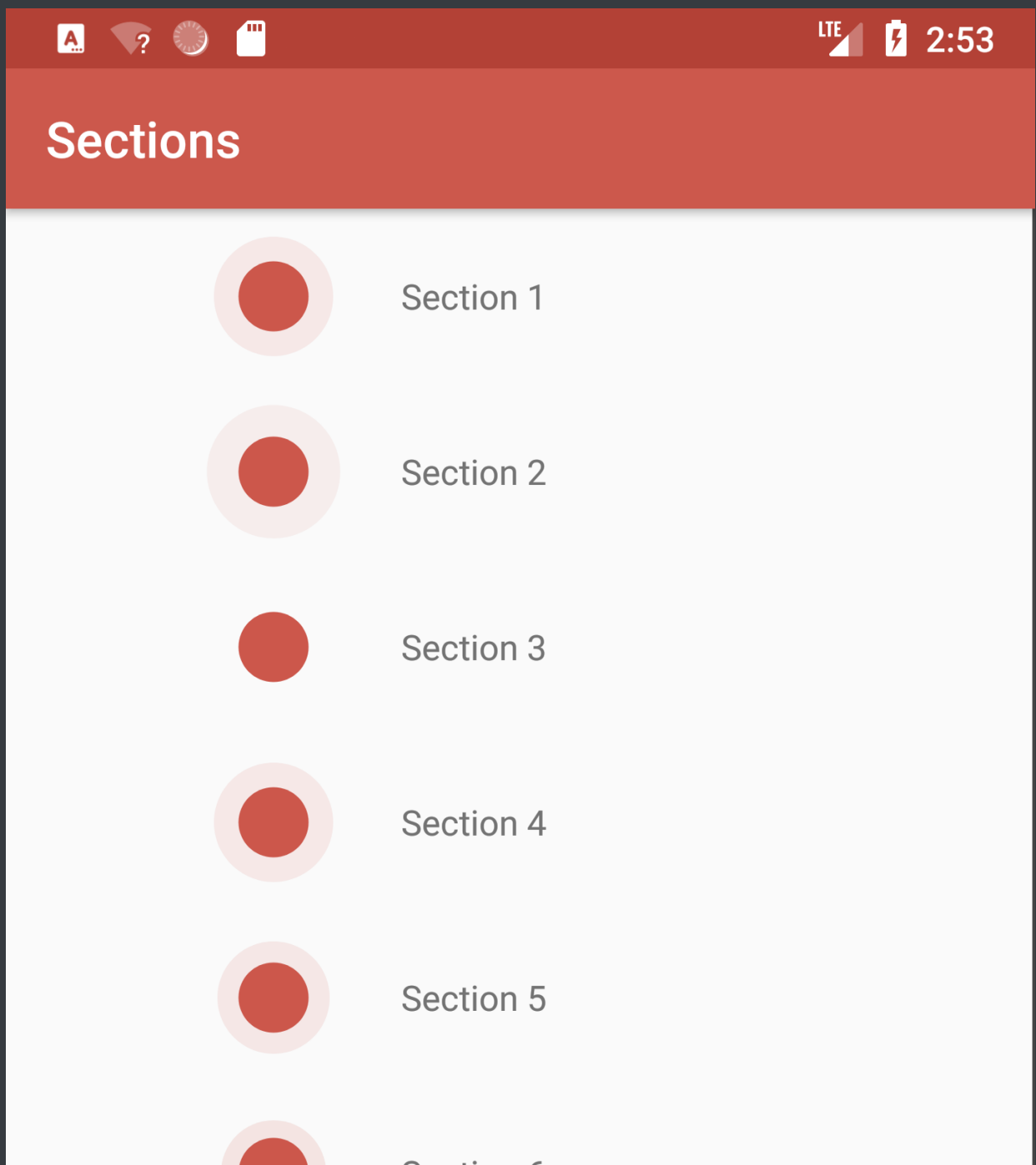


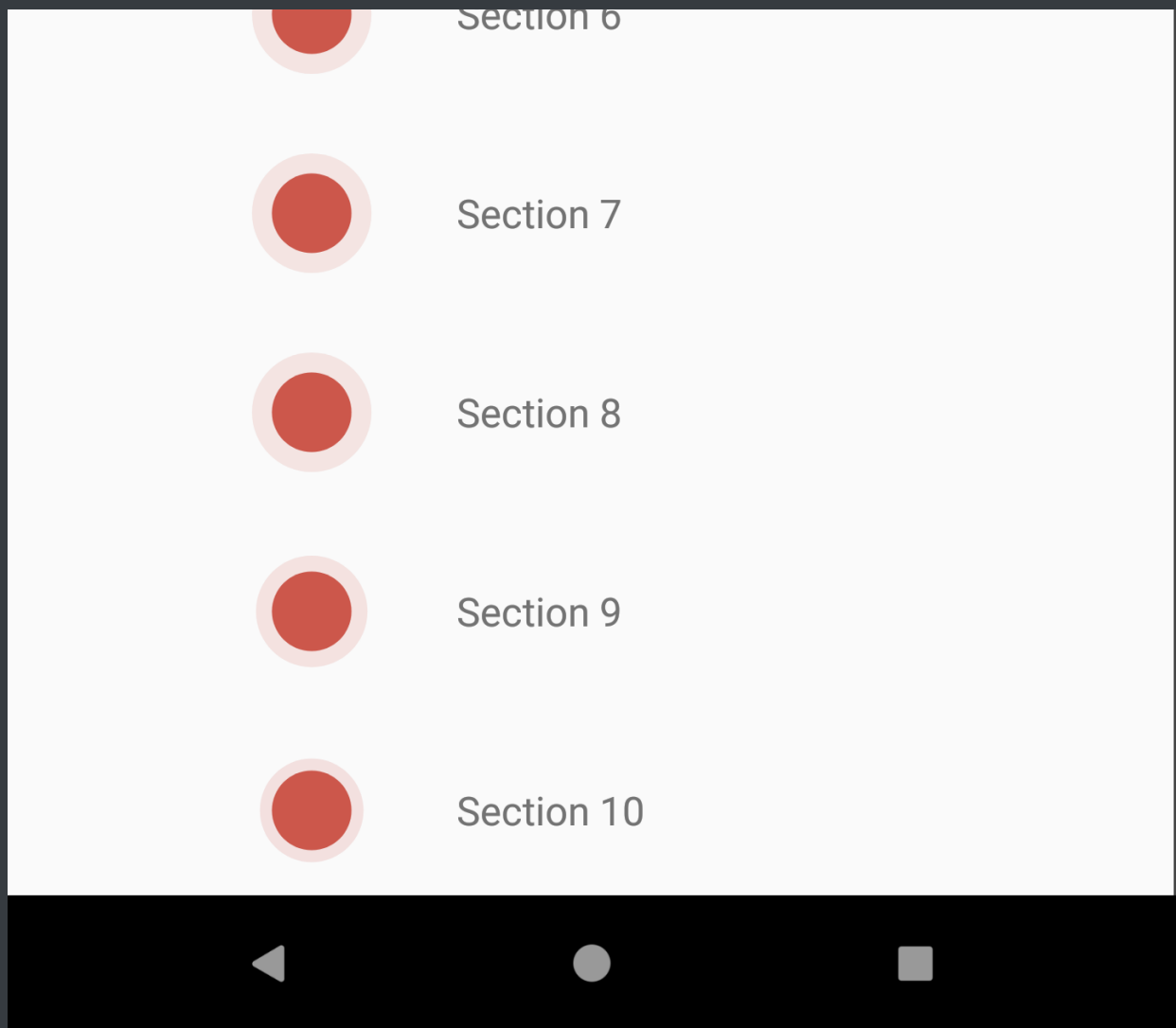
Cool chapter 3
we going to learn a lot!



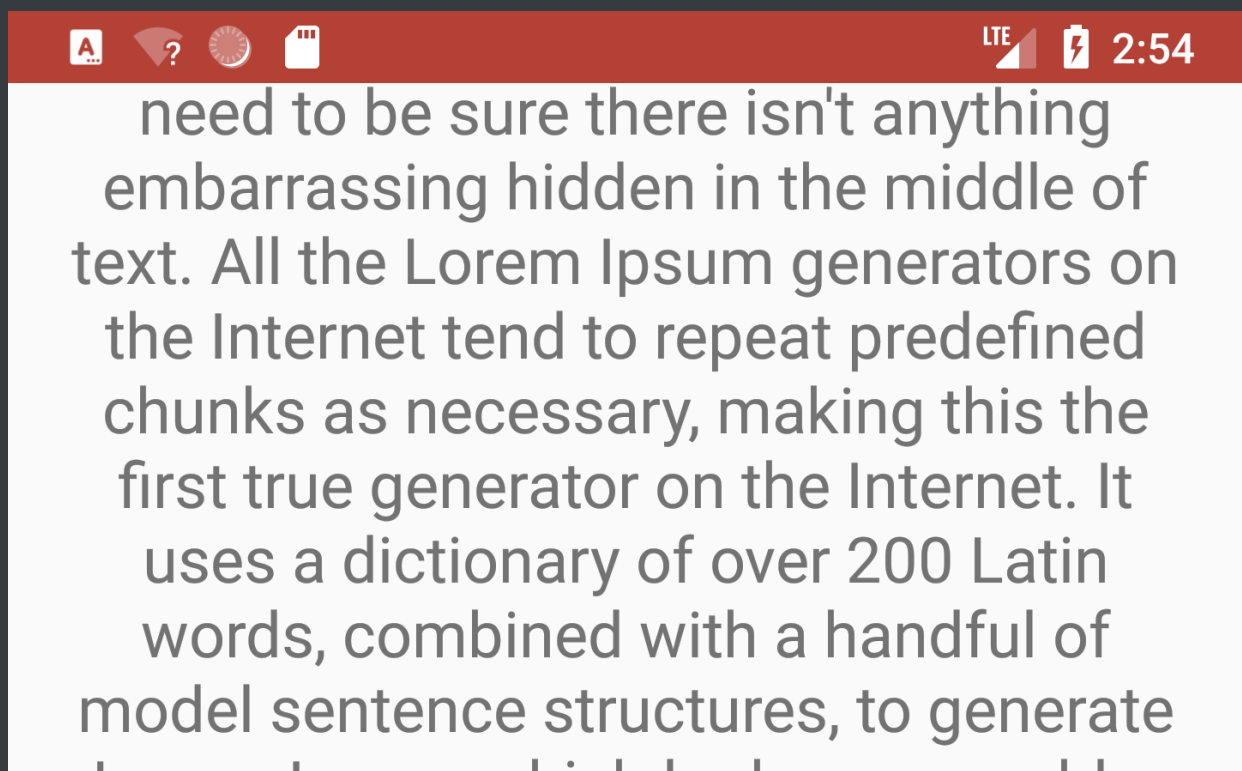


Section Screen





Section Contents Screen



Lorem Ipsum which looks reasonable.
The generated Lorem Ipsum is therefore
always free from repetition, injected
humour, or non-characteristic words etc.



What is the width of the security thread
in 1000 Rupee Note?

☐ 1 mm

☐ 2 mm

Server Implementation

Overview

The Back-end is built on top of the Spring-Boot 2.0 framework.

Rational

We've used Spring because it's very easy to setup and use, as well as being a very well documented framework.

Architecture Walkthrough

Models

Data transfer objects - these are encoded into JSON strong and are used to communicate with the client.

Domain

ORM objects - used to communicate with the database via Hibernate.

Mapper

Converter/adaptor between the domain and model objects.

Controllers

REST API definitions.

Repositories

Data access objects - database ORM communication definitions

Services

These are the business logic layer - used for managing and mediating between the controllers and database.

Implementation Details

Network

REST API on top of Tomcat server. Data is passed by way of POST and GET requests. Sensitive data is encrypted with RSA public key (per session) at the client side. Calls must contain a valid authentication key in order to get a response.

Database

Postgres database was used in combination with Hibernate ORM library (JPA based).

Encryption

We've used public-private key encryption (RSA) to secure client-server communication. The JWT library was used for this purpose.

Known Shortcomings

1. Naive security
2. Not passed any unit and integration testing

Potential Improvements

1. add user progress and status updates (book completion, points, last read, etc).
2. chat
3. microservices

Client Implementation

Overview

1. The client Android app is implemented for min-sdk 23.
2. The app uses a MVP (Model, View, Presenter) type architecture to separate between user interface and logic layers.

Rationale

1. A high min-sdk level was chosen to minimize development hurdles by enabling use of improved APIs (e.g vector graphics).
2. MVP was chosen after 2 mid-project re-writes. It allows for a good measure of layer separation without leaning towards over-engineering.

Architecture Walkthrough

Activities

Referred to as Views in the app. These are used for UI control and framework mediation.

Presenters

Presentation and business logic containers.

Contracts

Interface objects which define the relationship between a presenter and

Models

Domain layer containers - they're peculiar in this project because they're declared as abstract. Initially they were meant for server/client reusability, which didn't pan out.

Entities

Domain layer implementation.

Gateways

Used to access the remote server and local database.

Database

```
{
  "formatVersion": 1,
  "database": {
    "version": 1,
    "identityHash":
"dc9ba407b634d5c90f6c4d5f5436776e",
    "entities": [
      {
        "tableName": "Book",
        "createSql": "CREATE TABLE IF NOT EXISTS
`${TABLE_NAME}` (`bookId` INTEGER NOT NULL,
`coverUrl` TEXT, `thumbnailUrl` TEXT, `title` TEXT,
`tagLine` TEXT, `description` TEXT, `publishedDate`
INTEGER NOT NULL, `revisionDate` INTEGER NOT NULL,
`version` INTEGER NOT NULL, PRIMARY KEY(`bookId`))",
        "fields": [
          {
            "fieldPath": "bookId",
            "columnName": "bookId",
            "affinity": "INTEGER",
            "notNull": true
          },
          {
            "fieldPath": "coverUrl",
```

```
        "columnName": "coverUrl",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "thumbnailUrl",
        "columnName": "thumbnailUrl",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "title",
        "columnName": "title",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "tagLine",
        "columnName": "tagLine",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "description",
        "columnName": "description",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "publishedDate",
        "columnName": "publishedDate",
        "affinity": "INTEGER",
```

```
        "notNull": true
    },
    {
        "fieldPath": "revisionDate",
        "columnName": "revisionDate",
        "affinity": "INTEGER",
        "notNull": true
    },
    {
        "fieldPath": "version",
        "columnName": "version",
        "affinity": "INTEGER",
        "notNull": true
    }
],
"primaryKey": {
    "columnNames": [
        "bookId"
    ],
    "autoGenerate": false
},
"indices": [],
"foreignKeys": []
},
{
    "tableName": "Chapter",
```



```
    "createSql": "CREATE TABLE IF NOT EXISTS
`${TABLE_NAME}` (`chapterId` INTEGER NOT NULL,
`bookId` INTEGER NOT NULL, `chapterNum` INTEGER NOT
NULL, `name` TEXT, `description` TEXT,
`thumbnailUrl` TEXT, PRIMARY KEY(`chapterId`),
FOREIGN KEY(`chapterId`) REFERENCES `Book`(`bookId`)
ON UPDATE NO ACTION ON DELETE CASCADE )",
    "fields": [
        {
            "fieldPath": "chapterId",
            "columnName": "chapterId",
            "affinity": "INTEGER",
            "notNull": true
        },
        {
            "fieldPath": "bookId",
            "columnName": "bookId",
            "affinity": "INTEGER",
            "notNull": true
        },
        {
            "fieldPath": "chapterNum",
            "columnName": "chapterNum",
            "affinity": "INTEGER",
            "notNull": true
        },
        {
            "fieldPath": "name",
            "columnName": "name",
            "affinity": "TEXT",
            "notNull": false
        }
    ],
```

```
{
  "fieldPath": "description",
  "columnName": "description",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "thumbnailUrl",
  "columnName": "thumbnailUrl",
  "affinity": "TEXT",
  "notNull": false
}
],
"primaryKey": {
  "columnNames": [
    "chapterId"
  ],
  "autoGenerate": false
},
"indices": [],
"foreignKeys": [
  {
    "table": "Book",
    "onDelete": "CASCADE",
    "onUpdate": "NO ACTION",
    "columns": [
      "chapterId"
    ],
    "referencedColumns": [
      "bookId"
    ]
  }
]
```

```

    ]
  },
  {
    "tableName": "Section",
    "createSql": "CREATE TABLE IF NOT EXISTS
`${TABLE_NAME}` (`sectionId` INTEGER NOT NULL,
`chapterId` INTEGER NOT NULL, `sectionNum` INTEGER
NOT NULL, `name` TEXT, `explanation` TEXT,
`imageUrl` TEXT, `question` TEXT, `wrongAnswer1`
TEXT, `wrongAnswer2` TEXT, `wrongAnswer3` TEXT,
`correctAnswer` TEXT, PRIMARY KEY(`sectionId`),
FOREIGN KEY(`sectionId`) REFERENCES
`Chapter`(`chapterId`) ON UPDATE NO ACTION ON DELETE
CASCADE )",
    "fields": [
      {
        "fieldPath": "sectionId",
        "columnName": "sectionId",
        "affinity": "INTEGER",
        "notNull": true
      },
      {
        "fieldPath": "chapterId",
        "columnName": "chapterId",
        "affinity": "INTEGER",
        "notNull": true
      },
      {
        "fieldPath": "sectionNum",
        "columnName": "sectionNum",
        "affinity": "INTEGER",
        "notNull": true
      }
    ]
  }
]

```

```
},
{
  "fieldPath": "name",
  "columnName": "name",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "explanation",
  "columnName": "explanation",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "imageUrl",
  "columnName": "imageUrl",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "question",
  "columnName": "question",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "wrongAnswer1",
  "columnName": "wrongAnswer1",
  "affinity": "TEXT",
  "notNull": false
},
{
  "fieldPath": "wrongAnswer2",
  "columnName": "wrongAnswer2",
  "affinity": "TEXT",
  "notNull": false
}
```

```
        "fieldPath": "wrongAnswer2",
        "columnName": "wrongAnswer2",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "wrongAnswer3",
        "columnName": "wrongAnswer3",
        "affinity": "TEXT",
        "notNull": false
    },
    {
        "fieldPath": "correctAnswer",
        "columnName": "correctAnswer",
        "affinity": "TEXT",
        "notNull": false
    }
],
"primaryKey": {
    "columnNames": [
        "sectionId"
    ],
    "autoGenerate": false
},
"indices": [],
"foreignKeys": [
    {
        "table": "Chapter",
        "onDelete": "CASCADE",
        "onUpdate": "NO ACTION",
        "columns": [
            "sectionId"
```

```
    ],
    "referencedColumns": [
        "chapterId"
    ]
}
]
}
],
"setupQueries": [
    "CREATE TABLE IF NOT EXISTS room_master_table
(id INTEGER PRIMARY KEY,identity_hash TEXT)",
    "INSERT OR REPLACE INTO room_master_table
(id,identity_hash) VALUES(42,
\"dc9ba407b634d5c90f6c4d5f5436776e\")"
]
}
}
```