**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## **4.1** Проектирование программ сверху вниз

Самое главное условие успешного создания как крупного, так и небольшого приложения заключается в применении надежных методов проектирования.

Самое широкое распространение при написании такого рода программ получили следующие методы:

* нисходящий (сверху-вниз);
* восходящий (снизу-вверх);
* специальный (на данный конкретный случай).

В случае нисходящего метода процесс разработки программы начинается с программы высокого уровня и спускается до программ низкого уровня.

В свою очередь восходящий метод работает в обратном направлении, то есть сначала идет разработка отдельных специальных подпрограмм, на основании которых в последствии строятся более сложные, которые заканчиваются самым верхним уровнем приложения.

Специальный подход не имеет какого-то заранее установленного способа разработки, то есть решения в выборе метода происходят во время процесса разработки и могут меняться в любой момент времени.

JavaScript является, в одно время, как структурированным так и не структурированным языком. Именно поэтому в качестве разработки был выбран язык TypeScript. Он представляет собой строгий и структурированный JavaScript и лучше всего подходит для нисходящего программирования. Нисходящий подход позволяет писать ясный, легко читаемый код, что в дальнейшем не вызовет проблем с сопровождением приложения. К тому же данный метод отлично подходит для прояснения и создания всей структуры программы в целом до кодирования более низких уровней приложения.

Выбранный метод позволяет уменьшить потери времени, обусловленные неудачными и ошибочными решениями на начальном этапе. А в условиях небольшого срока разработки программы – это делает его основным методом разработки и позволяет перейти к проектированию программы.

## **4.2** Структурирование программы

Как описывалось выше, для любой общей схемы при применении метода «сверху-вниз» разработку начинают с общего описания приложения, а после этого двигаются в направлении проработки ее конкретных деталей, компонентов и прочего.

При разработке любой программы лучше всего точно определить, что и как программа будет делать на самом высоком уровне, а затем уже погружаться в детали, касающиеся каждого ее действия. На рисунке 4.1 представлена схема нисходящего метода разработки.



Рисунок 4.1 – Схема нисходящего метода разработки программы

Исходя из изображения выше можно заметить, что эта схема очень легко ложиться на структурную схему разрабатываемого приложения, что говорит о правильном выборе метода разработки. Согласно схеме, можно легко определить и выделить два основных модуля программы. Модулей два потому как приложение содержит клиентскую и серверную части, каждая из которых служит обособленным модулем высокого уровня. К модулям высокого уровня относятся:

* блок ядра клиентского приложения;
* блок ядра серверного приложения.

Теперь можно приступать к составлению перечня действий, которые будет выполнять каждый отдельный модуль.

## **4.3** Блок ядра серверного приложения

Разработка начинается с ядра серверного приложения потому как разрабатывать клиентское ядро без минимально функционирующей серверной части не имеет смысла.

Первая и самая основная цель серверной стороны – это запускать сервер. Так как сервер будет работать на платформе NodeJS, то самый удобный и безопасный способ работать с сервером – это подключить библиотеку ExpressJS. Это уже давно зарекомендовавший себя модуль, который имеет большой и гибкий набор конфигураций.

Необходимо добавить данную библиотеку в проект с помощью NPM-менеджера. Для этого нужно использовать команду: «npm install express --save». После установки нужно подключить модуль к основному файлу сервера и произвести пробный запуск сервера на 1337 порту. По умолчанию, сервер будет запущен на локальном адресе компьютера без каких-либо дополнительных настроек. Пробный запуск сервера представлен на рисунке 4.2.



Рисунок 4.2 – Тестовый запуск сервера на ExpressJS модуле

После успешного тестового запуска сервера можно приступить к настройке его конфигураций.

Имеет смысл вынести конфигурацию и настройки сервера в отдельный файл. Это упростит работу с проектом и дальнейшее ориентирование в файлах проекта. Сервер должен быть достаточно безопасным, так как будет хранить данные бюджета пользователей. Здесь будет использоваться подход, при котором к серверу будет открыто как можно меньше путей. Именно поэтому в конфигурации сервера нужно закрыть доступ ко всем скрытым файлам, начинающимся с точки. Также в конфигурации слудет указать, чтобы по умолчанию открывались только файлы с расширением html и не разрешалась никакая переадресация.

Сервер будет содержать и статический данные, такие как скрипты, файлы стилей и скриптов. Имеет смысл перенести статичесикие файлы в одну папку и запусть сервер для работы именно с ней в качестве корневого каталога. Таким образом доступ к серверу будет строго ограничен статическими файлами.

Для обмена данными между клиентом и серевером будет использоваться формат JSON – это текстовый формат обмена данными, основанный на JavaScript. По умолчанию данный модуль не включен в конфигурацию сервера, поэтому его необходимо добавить через NPM-менеджер, а затем добавить в конфигурацию.

Постоянно загружать статические файлы по запросу клиента не имеет смысла, именно поэтому в конфигурацию также включается настройка кеша. В настройке нудно указать максимыльный срок кеширования файлов, в данном случае это будет один день. А также необходимо включить использование eTag – часть HTTP. Это один из нескольких механизмов, с помощью которых HTTP обеспечивает веб-проверку кэша и который позволяет клиенту делать условный запрос. Это позволяет кэшу быть более эффективным и экономит пропускную способность, так как веб-серверу не нужно отправлять полный ответ, если содержимое не изменилось. ETag также может быть использован для оптимального управления многопоточностью как способ, чтобы помочь предотвратить одновременное обновление и перезапись ресурса.

Etag – это закрытый идентификатор, присвоенный веб-сервером на определенную версию ресурса, найденного на URL. Если содержание ресурса для этого адреса меняется на новое, назначается и новый ETag. Использование в таком ключе ETags аналогично использованию отпечатков пальцев, можно быстро сравнить и определить, являются ли две версии ресурса одинаковыми или нет. Сравнение ETag имеет смысл только c Etag с одного и того же URL, идентификаторы, полученные из разных URL-адресов, могут быть, а могут не быть равны, вне зависимости от ресурсов, так что их сравнение не имеет какого-либо смысла.

Также для кеширования будет использоваться свойство заголовка «Last modified». Данный заголовок также служит в качестве элемента кеширования данных. В нем содержится дата последнего изменени файла и если при запросе, клинетские браузер видит, что дата изменения не менялась, то подгружает файл из кеша браузера.

На данном этапе настройка конфигурации сервера заканчивается. Теперь сервер будет раздавать статический файлы, а значит можно заняться разарботкой статических файлов на клиентской стороне.

## **4.4** Блок ядра клиентского приложения

Основное ядро клиентской части приложения будет содержаться в JavaScript файле. Однако его нужно загрузить в HTML файл. HTML файл будет статически загружаться с сервера, автоматически загуражать статические стили и основной JavaScript файл, после чего начнеться посроение всех частей приложения из основного ядра.

В HTML файлe указываются пути для загрузки скриптов и стилей, также указываются мета-теги для оптимизации страницы браузера под мобильные устройства. Как описывалось ранее, в дальнейшем приложение будет обернуто в прилодение Cordova. Следовательно функции зумирования страницы должны быть отключены.

В теле страницы описываются основные элементы страницы, на основании которых будут строиться остальные части программы. К данным элементам относяться:

* блок заголовка;
* блок меню;
* блок основных элементов.